DICK SMITH
VZ200
Personal
Colour Computer

Introduction
to Computing

B-7200

Introduction to Computing

Using your VZ-200
Personal Colour Computer!

Written and illustrated
by Toni Louise Henson

FIRST EDITION, 1983

Table of Contents

Chapter 1
The Ancestry of a Computer

Isn't it exciting? You've just brought your brand new VZ-200 personal computer home. And naturally the first thing you want to know is how to use it. Well, we'll get started on that in just a moment. But first, let's make sure you know exactly what that new machine of yours is!

Everyone knows that computers are electronic brains, right?

   No, wrong!
They don't really have brains at all.

Forget about whatever you've seen in movies...amazing machines that hold intelligent conversations, have ideas or plot to take over the world. A computer is more like a trained parrot...you can teach it to do tricks and obey commands, and even "talk" to you. But it certainly can't think like you and I!

That's what a computer isn't. To explain exactly what a computer is, let's take a quick look at your little VZ-200's ancestry.

From the beginning of history, man has been looking around for ways to make his life easier. He discovered that there were limits to what his body could do...and so he began inventing ways to extend his physical capabilities.

The caveman is a good example. At some stage he discovered that he could hit something much harder if he used a rock instead of his bare hands. Which made the hitting process easier (and less painful!).

As the ages passed, man invented a multitude of "helpers" to extend his body's limited abilities. When he eventually learned to talk, he gave these helpers a name: he called them "tools".

Today, humans rely on many tools. A car, for example, is a tool to help move us along much faster than our legs can.

A telephone is a tool to help us talk to someone much further away than our voices can shout.

A microscope is a tool to help us see objects far smaller

than our eyes can detect without its help.

And a computer is a tool to help us remember much more, and do calculations must faster, than our brain can manage on its own.
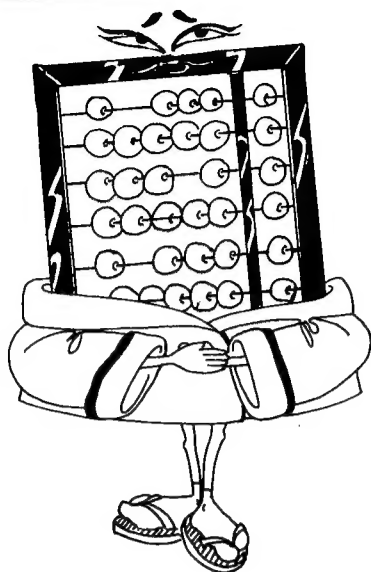
Remember the caveman and his hammer? Over the years his descendants improved on the rock idea...by adding a handle, changing the shape, using metal instead of stone.

Our modern aids evolved in pretty much the same way. In the case of the computer, it started way back with the ancient Chinese abacus!

What we've really been trying to say is this: Don't ever be scared of your VZ-200!

Just remember that it's nothing but a tool to extend your brain's abilities! It can't think without your help; it hasn't any intelligence of its own; and it's certainly not going to laugh at you if you do something wrong!

And while we're doing some remembering, here's another point that's well worth bearing in mind:

No matter what you put into your little computer, you can't hurt it! You can play around with it, type in as much nonsense as you like...but you'll never, never get yourself into a situation that you can't get out of. 'Cause if worst comes to worst, all you have to do is turn it off. And start again! So please, feel free to experiment. Why wonder what would happen if you pressed that button...when you can try it and see?!

Chapter 2

Bringing your VZ-200 to life!

To connect up your computer, you'll have to turn your television set off. (Oh yes...if someone happens to be watching it, you'd better ask first!)

At the back of the television, you'll find a cable that connects the set to the aerial. This is how information from a television station is received.

(BACK)
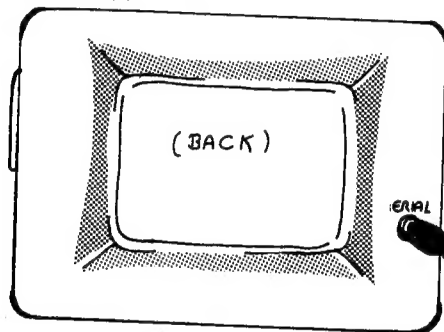
ERIAL

Just at the moment, though, we want your telly to receive information from the VZ-200! So: Disconnect the aerial cable from its socket. Now, replace it with the thick, single cord (it's in the box your computer came in). The fattest end of this cord goes in the hole.

Well, that takes care of one end of the cable! The other end goes into the furthest right-hand socket at the back of the VZ-200. (If you look closely, you'll see "TV" written beneath the hole!)
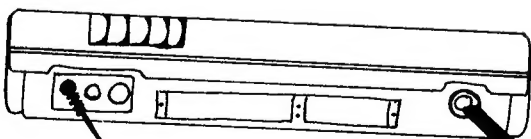
In your computer's box, there should also be a black box with a thin cord attached. This very strange-looking device is called a "power adaptor"... because it "adapts" the electric current from an ordinary power-point, to suit the VZ-200's needs. (Your computer, you see, is rather fussy about the sort of power that you feed it!)

First, plug the black box into a power-point. Done that? Right...now, at the end of that long, thin cable you'll find a little plug. Push it into the furthest left-hand socket at the back of your VZ-200. (This hole has writing under it too -- it says "DC9V"
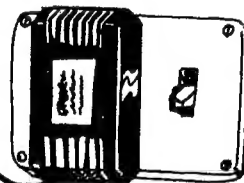
Turn your computer on by pressing the switch on the left-hand side. (If everything's ok, the little red power light on the top of the VZ-200 should begin to glow.)
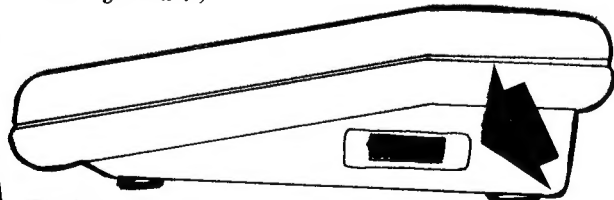
Switch your telly on once more. And turn the channel selector to Channel 1.

That's all!

FINISHED ALREADY!

Now, everything's all set up for operation. And look at that! If you've done everything just right, your VZ-200 will be talking to you already...saying it's READY to listen to whatever you tell it.

You should also be able to see the cursor. No, that's not someone who swears a lot! It's the flashing green square on the third line, and it's there to tell you exactly where you are on the screen.

If your VZ-200 isn't giving you a READY message, don't worry. Go back and make sure the connections aren't loose, or that the cords are in the right sockets.

ONE BIG DIFFERENCE BETWEEN COMPUTERS & HUMANS

When human beings talk to one another, they need to use two things:

> Ears to hear what's being said, and
> A Mouth to give answers, ask questions and so on.

Take a look at your VZ-200. You certainly won't see a pair of ears and a mouth like yours...but computers do have things to do much the same job!

The VZ-200's keys are its "ears". When you type in questions or instructions, it "hears" you. And when the VZ-200 wants to talk back to you, its "mouth" is the display screen (in this case, your TV set!).

Can you guess what this means? Right--to hold computer conversations, you'll have to use your senses a little differently to the way you usually do.

In other words, when you talk to a computer your fingers become your voice, and your eyes become your ears!

"BUT HOW DO I USE THE KEYBOARD?"

If at this stage all those keys on your VZ-200 look a little confusing, don't worry. Firstly, let's call the whole section where the keys are "the keyboard". Secondly, let's find out how simple keyboards are to understand!

There are a whole lot of letters, numerals, symbols, commands and so on that your computer understands. Of course, we could have given each one of these a key all to itself. But that would have made the VZ-200's keyboard very, very crowded!

We thought it was a much better idea to give some keys more than one job. These are called "multi-function" keys...and they may be able to do as many as four things!

To show you exactly how this works, let's pick any old key. "P" will do nicely. On the keyboard it looks something like this:

As you can see, there are two symbols on the key itself.
Want the letter "P"? Easy--just press the P key!
If you want the "]" symbol, first find the key
marked SHIFT. (It's in the bottom left-hand corner
of the keyboard). Got it? Now, hold it down while
you press the P key.

There are two more commands written above and below the P
key.
To get "PRINT ", hold down the CTRL key (it's right
above SHIFT) while you press P.
If it's "NOT" you're after, there's two things to
remember: 1. Hold down CTRL while you press the key
marked RETURN . Then let go of the RETURN key!
2. Now type P.

The same rules work for every key with more than one job to
do. Just remember that
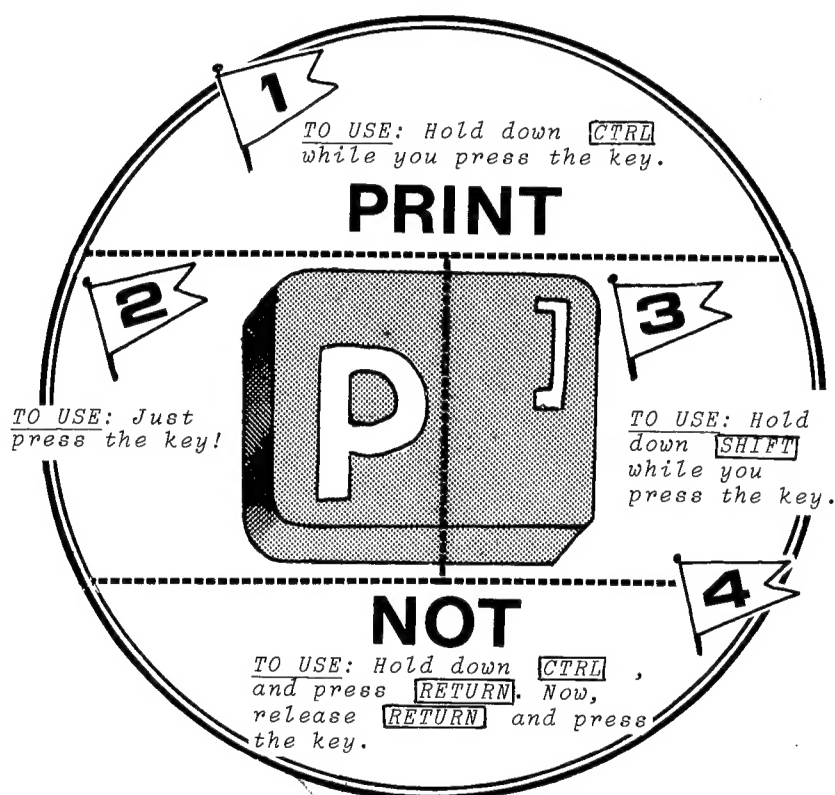
(Oh, by the way--those funny box-shaped symbols on some of
the keys are called "graphics characters". We'll be using
them later, when we learn to draw pictures on the screen..)

Chapter 3

Programming in a Nutshell!

"Computer Programming", we say.

"Help!" you think. Well, that's a very natural reaction to something you don't know much about!

Wait! Good news! It's not nearly so complex as you've always thought...in fact, it's really very, very easy to understand. Especially if you imagine it like this:

Suppose you're about to start a dot-to-dot picture puzzle (we all know how easy they are!). You know that all you have to do is start at the first step (that's the dot labelled "1"). Then move your pen to dot number two, then dot number three, and so on until you've reached the last number in the puzzle.
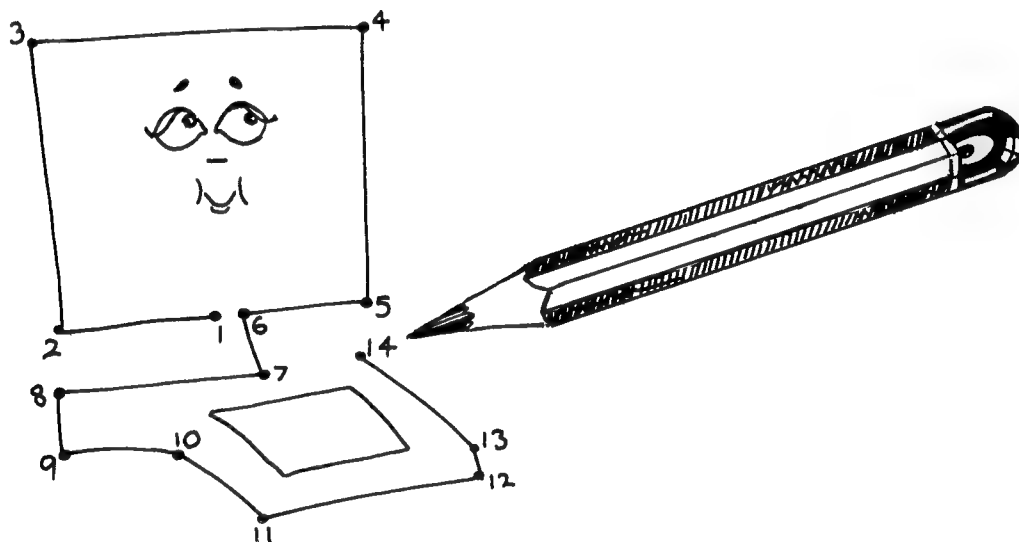
Nothing hard about that, is there? All you've been doing is following a set of steps that somebody has designed for you. And by following those steps you created a picture--even though you didn't know what it would look like. In other words, you followed a program!

So! When you become a programmer, all you're doing is planning a set of steps for your VZ-200 to follow.

There's something very important to remember here. What do you think would happen if the points in that dot-to-dot puzzle were numbered in the wrong order...or if some dots were missed out altogether? Can you guess what you'd end up with? Right--a picture that made no sense at all!

To get your VZ-200 to make sense of the program you give it, just be sure to tell it each step in the correct order.
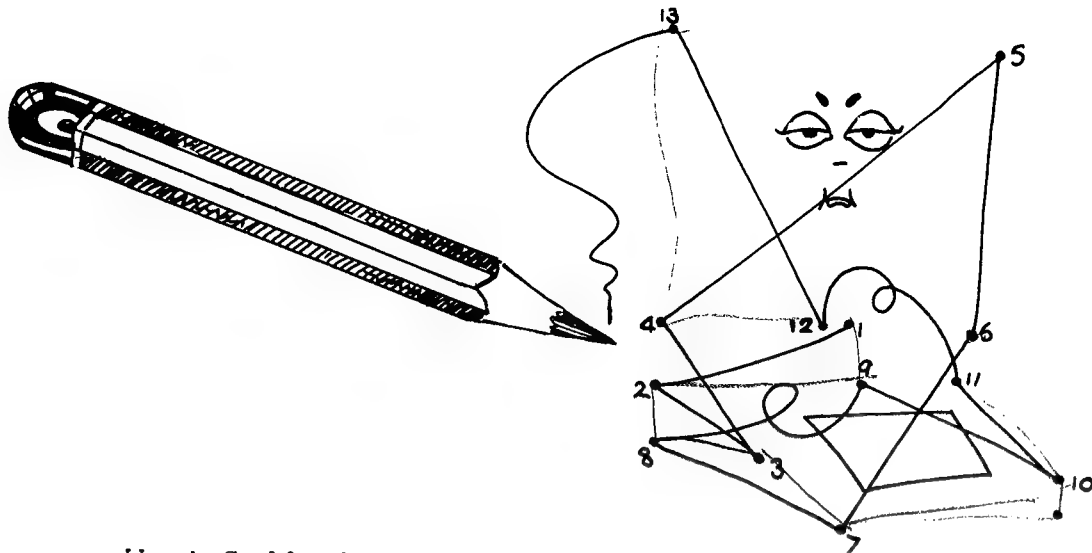
Wow! Suddenly we know a whole lot more about that
mysterious thing called "computer programming."
(Doesn't sound so scary now, does it?)

In a couple of minutes flat, we've learnt two
things about getting your VZ-200 to do just what
you want:

- You need to tell it the <u>steps</u> it must follow
  in order to do the job...
- And the <u>sequence</u> in which it must take those
  steps

Terrific! But hang on...there's one thing we'll need to
know before we can tell the VZ-200 anything. And that's how
to talk to it!

BREAKING THE LANGUAGE BARRIER

Remember what we told you last chapter about the
differences between computers and humans? Here's another
one -- computers aren't smart enough to understand the
complex language we use!
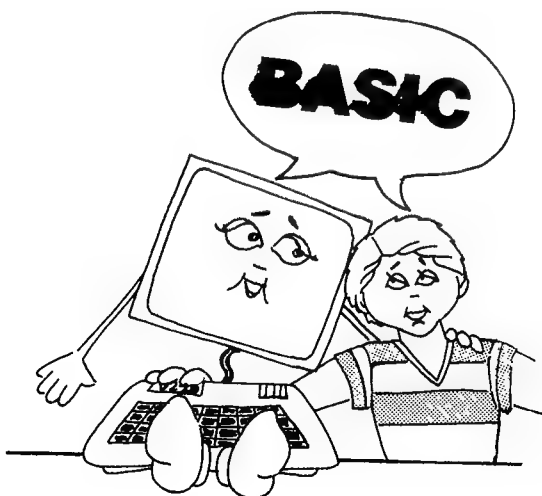
To overcome the "language
barrier", humans speak to
computers in a special language
called "BASIC".

You'd probably expect something
with a name like BASIC to be very
easy. And that's exactly what it
is! In fact, it's a whole lot like
English...but with far fewer
words.

You can order a fellow human to perform thousands of duties...everything from "Make me a cheese sandwich", to "Go and climb that flagpole"! Naturally, we need many, many words to give instructions for these different duties. A computer, on the other hand, is capable of performing relatively few duties. And because of this, there are relatively few words that it needs to understand! (Just try asking your VZ-200 to make a sandwich for you...it won't know what you're talking about!)

AND NOW FOR A BIT OF COMPUTER PSYCHOLOGY...

As we've mentioned, a computer isn't an electronic brain. In fact, it doesn't have a brain at all! It's really only capable of REMEMBERING (using its memory) and OBEYING ORDERS (using its processor).

Human beings, on the other hand, are equipped with brains...and we can use them to think, have ideas, work out problems, show initiative, and all sorts of clever things.

So, when you're trying to hold a "conversation" with your VZ-200, it's important to remember that difference. Because your VZ-200 will happily do exactly as it is told...and do it very, very quickly. But it won't do anything unless you tell it to!

That's where you could run into a little trouble! After all, most human beings are used to talking to other humans...and it's very easy to simply forget how silly your computer really is.

Here's an example: suppose you said to a friend :"Add two and two". They'd think for a moment (how long depends on how bright your friend is!), then tell you the answer. Why? Because they'd automatically assume that you wanted to know it.

But if you asked your VZ-200 the same question, you could wait all day. Your computer won't tell you the answer until you tell it to! That's because unlike your human friend, it's just not capable of making an assumption.

Your computer will never understand what you mean if you don't make yourself clear.

If instance, suppose you said to a human friend: "Go and get me my measuring thingumy. It's in one of the drawers over there." They'd probably work out that you wanted your ruler, and look through all the drawers until they found it.

But if your friend's brain worked like a computer, they simply wouldn't be able to work out what you wanted! Instead they'd say: "I don't understand! What is a measuring thingumy? How do I get to the drawers? Which drawer do I look in? Or they might just sit there and look at you blankly!

Just remember this: computers mightn't be as smart as

us...but they very rarely make mistakes! So, exactly what can you do if a program does something quite different to what you wanted? Or even refuses to do anything at all? Well, here's what you _don't_ do: please _don't_ call the VZ-200 rude names. Not because it's easy to hurt its feelings...but because your computer probably isn't at fault!

999,999 times out of a million, things will be going wrong because you--the programmer--have made a boo-boo.

So this is what you must do: make sure that you've told your VZ-200 _every single thing_ that you want it to do. Because if you've accidentally left something out, the computer will never guess what it is!

Chapter 4

Let's Get Started!

Right! Now that you know such a lot about computers, we're going to do a bit of programming. (Hey! You're not still scared of that word, are you? We've already told you how simple it is!)

Actually, programming your VZ-200 is as simple as saying "Hello". And to prove it, "Hello" is the very first thing we'll teach it to say!

USING LINE NUMBERS

Remember the dot-to-dot puzzle? Every dot had a number. And those numbers told you the order in which to take each step.

Your VZ-200 needs to know the order in which to take the steps in a program. So we'll give every step (or line) a number, too.

You could just number the lines 1, 2, 3 and so on...but it's usually not a good idea. When you start to write longer programs, you just might leave out a line or two by accident. (It happens to everyone occasionally!) And if your lines are numbered 1, 2, 3, etc, you'll have no spare line numbers to give your forgotten steps.

It's a better idea to give your lines numbers with gaps between them--like 10, 20, 30 and so on. That gives you 9 spare line numbers between each step in your program, just in case.

OK, VZ-200--SAY "HELLO"!

Type the lines inside the box below, exactly as they appear there. (What's that? Did you say you're not sure how? OK--just take a look at them for now. We'll go through it key by key, to show you how simple it is.)

```
10 PRINT "HELLO" [RETURN]
20 GOTO 10 [RETURN]
```

Right, now the first thing we want to put on the top line is the number 10 (that's the line number we told you about in chapter 3). Simply press the key marked 1, and then the key marked 0. And look! There's your 10 on the screen. And the little flashing cursor is now directly after the 0...marking the spot where the next thing you type will appear.

(Incidentally, did you hear a 'beep' each time you pressed a key? That's the VZ-200's way of saying "Yep--I heard you.")

What comes next in the first line of our program? It's the word PRINT, isn't it? Well, no...not quite. There's something that has to go in front of it--a space! To put a space between things on the VZ-200's screen, use the key marked SPACE (Can you see it? It's in the bottom right-hand corner of the keyboard.) Try it now--just press once, and release. Sure enough, the cursor will have hopped one space to the right.

Now it's PRINT's turn...and there are a couple of ways to do it. There's the common or garden-variety method of typing it in letter by letter -- and if you want to do it this way, it will work just fine. Or (to save yourself all that typing) you can take a short-cut. See what's written above the P key? The word PRINT! And of course, you already know how to get to it. (What's that? You've forgotten? Shame on you! Go back and take a look at the diagram at the end of chapter 2.). Let's try using the short-cut, by holding down the CTRL key while we type P.

Hey, now that's convenient! Just as quickly as your VZ-200 could say 'beep', PRINT appeared on the screen. It looks like magic--but it's just a way of making things easier. There are dozens of useful words that're just as easy to 'conjure up'...you'll find them above, or below, most of the keys on the keyboard.
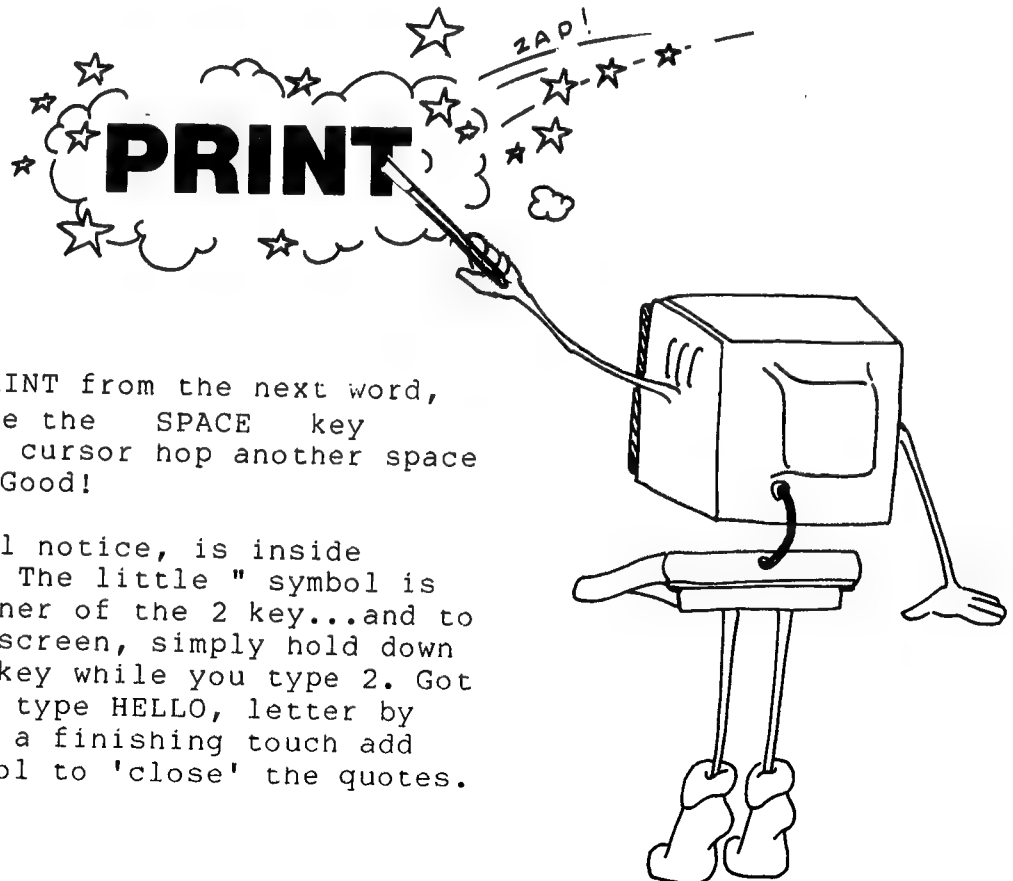
To separate PRINT from the next word, we'd better use the SPACE key again. Did the cursor hop another space to the right? Good!

"HELLO", you'll notice, is inside 'quote-marks'. The little " symbol is in the top corner of the 2 key...and to put it on the screen, simply hold down the SHIFT key while you type 2. Got it? Right--now type HELLO, letter by letter; and as a finishing touch add another " symbol to 'close' the quotes. Simple!

15

As the lucky last thing on your very first program line,
we've drawn a picture of the key marked ☐RETURN☐ . This
is really the easiest of all...just press it once and see
what happens. Goodness! Nothing new appeared on the
screen--but the cursor moved down to where the second will
begin!

Boy! What a lot of fuss over one little line! Now
that we've convinced you how simple this whole
business is, you can try your hand at typing in the
second line--all by yourself.

Type 20, then GOTO (it's just above the G key), then 10,
and press ☐RETURN☐ .

Do you know what you've just done? It's really pretty
exciting! You've "spoken" to your little computer, using
the keyboard. And you've used two words of the special
BASIC language we told you about in chapter 3.

Let's translate, and show you exactly what you said.

PRINT means "Write this on the screen". (Don't
forget--quote-marks must go around whatever you
want 'printed'.)
GOTO means "Go to the line that I'm about to tell
you.". It's always followed by a line
number--because you have to tell the VZ-200 exactly
where you want it to go! (In this case, we wanted
it to go back to line number 10. So we said "GOTO
10". Get the idea?)

Pressing the ☐RETURN☐ key simply tells your
computer: That's all I want to put on this
program line. Now, look at the next one!
This key follows just about every line you type
into your VZ-200.

Congratulations! Your first program is now sitting right
there in your VZ-200's memory.

Terrific, you say. But how come nothing's happening? Well,
it will in a moment...just as soon as we've taught you a
couple more BASIC words.

The words you're about to learn are called
direct commands. They don't need to have a line
number (although if you give them one, the
computer will still obey them). When your
VZ-200 sees a direct command, it knows that you
want it to do something with your
program...straight away!

The first one is LIST. This means "Show me the program
exactly as I typed it in". When you type LIST, your
program appears on the screen. Try it now.

(Nothing happened? It's probably because you forgot to

press RETURN...until you do that, your VZ-200 will think that you haven't finished giving it your command yet.)

The second is RUN--which means "Do as the program tells you". When you tell your VZ-200 to RUN and then press RETURN , it'll start to carry out the program you've given it.

Ready to see what your little program will do? OK...type RUN RETURN and watch this!

Wow! If you hadn't already figured out what was going to happen, all those "Hellos" on the screen probably gave you quite a surprise. But it's really not so surprising when you look at how the program works:

Remember, the VZ-200 starts with the smallest line number and works its way up.
> The VZ-200 went to line 10 first. That line told it to print "HELLO"
> The VZ-200 then goes to the next line (number 20).
> Line 20 tells the VZ-200 to go back to line 10.
> Which starts the whole thing all over again!

This clever trick is called "looping". Why? Because a loop (or circle) doesn't have an end...and neither does this type of program! It's a bit like sending your VZ-200 on a endless merry-go-round ride.
> If your VZ-200 refused to say HELLO to you, don't take it personally. Just type LIST again (you haven't forgotten what it means already have you?), and check that the program on the screen is exactly the same as the example in this book.
> You may find a mistake. Also, check that none of the VZ-200's connection cables are loose.

"BUT HOW DO I STOP IT?"

While we've been talking, your VZ-200 has been "Helloing"--over and over again. And it will keep it up until the cows come home, unless we "break" the program.

Here's another BASIC command that'll do exactly that. It's called BREAK, and it means "Stop where you are!"

You'll see BREAK written above the very last key in the top row. To use it, just press the key while you hold down CTRL . And now, here's the surprising part: you don't have to press RETURN after BREAK! Commands that don't need RETURN ing are as rare as hen's teeth (rarer, in fact -- this is the only one in existence!)
That certainly does the trick! Your screen should now show the last of the HELLOs, followed by:
> BREAK IN LINE 10 (or BREAK IN LINE 20...it depends on where your computer was when you "put on the brakes"!)
If you really liked the Hellos, and want see some more, you can say CONT RETURN to the VZ-200. This is short for

"continue"--and it tells your computer to keep running the program you broke.

Of course, you can easily <u>give</u> your program an end by giving it another BASIC instruction called--you guessed it--**END**! A program to make your VZ-200 say "Hello" just once would look like this:

```
10 PRINT "HELLO"
20 END
```

So **END** means "That's the end of this program".

Ready to move on to something else? OK...let's learn two more BASIC messages that'll get rid of those Hellos: **CLS** and **NEW**.

      CLS is short for "Clear the Screen"--what it's called is what it does.

Try it now.

      CLS |RETURN|

See? The screen is cleared, and the VZ-200 is telling you that it's READY once again.

      NEW means "Forget that program". It clears the VZ-200's screen <u>and</u> its memory. A bit like giving your computer instant amnesia!

Go ahead and press

      NEW |RETURN|

The screen won't change...but if you now try to tell the VZ-200 to RUN its program, it will just tell you that it's READY again. Which means "Program? <u>What</u> program? I'm READY to do something new, now!""

PLEASE, COW... WHY WON'T YOU COME HOME?

BECAUSE I'M A <u>BULL</u>, MATE!

BASIC words we've learned so far:

```
PRINT ".....,"    Write whatever's in here "......" on the
                  screen.
GOTO              Go to this line (remember to put a line
                  number straight after this command!)

END               That's the end of this program.

LIST              List my program on the screen.

RUN               Carry out the program.

BREAK             Stop running the program!
CONT              Continue running the program.

CLS               Clear the Screen.
NEW               Forget the program.
```

(And don't forget -- press the [RETURN] key
at the end of each line to let your computer
know you've finished with the line!)
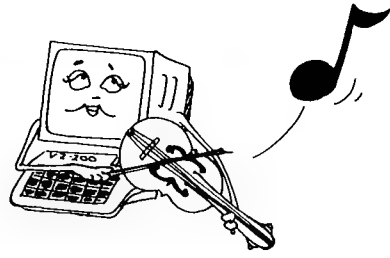
Notes

"OK...nobody's perfect!"

(Or, "How to fix Typing Mistakes!")

No matter how careful you are with your typing, sooner or later you'll find that you've pressed the wrong key. Don't despair...boo-boos are easy to make. You wouldn't be human if you didn't! And anyway, your VZ-200 is such a clever little computer that mistakes are almost as easy to fix, as they are to make!

If you look at the last four keys in the bottom row of the keyboard, what do you see? That's it...each key has a picture of an arrow above it. One arrow points to the left, one points to the right, one points up, and one points down.

> You can use these arrows to "steer" the cursor around the VZ-200' screen. The whole idea is to position the cursor directly over the mistake that you've made!

"Yes -- but how do they work?"

We were just coming to that. As we've mentioned, each of these little pointers is above a key. Which means that it's in section 1 of the diagram at the end of chapter 2.

> So! To operate any of the arrows, we know that we must first ...(all together, now!)...hold down the CTRL key!

Very good, class! If you hold down CTRL , every press of an arrow key will move the VZ-200's cursor one space in the direction of the pointer. (Up, down, left, right...it all depends on which arrow you choose!)

If you're not content to move just one space at a time, try leaving your finger on the arrow key. The cursor will keep moving across the screen (beeping all the way!) until you let go of the key!

A little warning: If you do try this trick, you may accidentally "overshoot" the place you were aiming for. Don't worry! Just use a different arrow key to move the cursor back to your target.
("Driving" the cursor round the screen is rather like driving a dodgem car...you can take it anywhere you like!)

A BIG warning: You must remember to hold down the CTRL key all the time , while using an arrow key. 'Cause if you don't, the cursor will leave a trail of destruction across your screen!
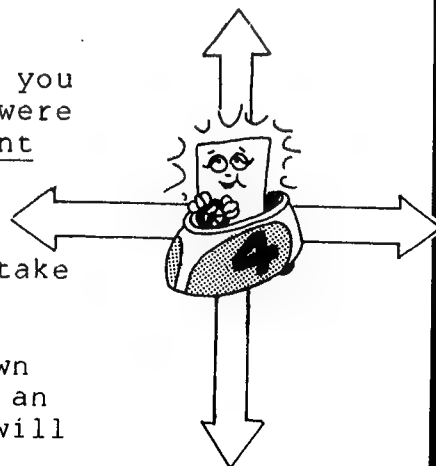
There are several ways of making boo-boos. Fortunately, there are also several ways of fixing them!

"Oops -- I've typed in an extra letter!"

Let's pretend that you typed line 10 like this:

        10 PRINT "HEELLO"

Hmmm..that looks a bit odd! We'll have to get rid of that extra "E" (unless, of course, you want to give your computer a Mexican accent!)
To "rub out" a letter (or a space, a number, or some other character) is simple...'cause there's a command to do the job for us. And it's called -- very logically -- RUBOUT! You'll find it above the ";" key.

To fix this mistake:

Using the arrows, position the cursor over one of the extra E's (it doesn't matter which). Now, hold down ⌷CTRL⌷ and press RUBOUT. The cursor will "swallow" the letter underneath it!

"Oh, rats! I've left out a letter"

Fortunately, there is also a command that allows a forgotten character to be "inserted" between others on the screen!
(Can you guess what it's called? Right -- INSERT! To find it, look above the L key.)

What if you had typed line 10 like this?:

        10 PRINT "ELLO"

That'll never do! We can't have a computer that "drops its H's"!

To fix a problem of this type:

Position the cursor over the character that should come after the one you've left out. (In this case, it would be the letter E)

When you hold down ⌷CTRL⌷ and press INSERT, the cursor will "push" the letter underneath it to the right. (Why? To make room for whatever character you wish to insert!) In the blank space that it's created, just type your forgotten H!

        Sorry -- you can't take shortcuts with INSERT!
        To insert more than one character, you must
        press INSERT before typing each one.

<u>"Good Grief! I've REALLY goofed this line up!"</u>

Now, we know that some people never do things by halves!
And occasionally, you might mess up a line completely.

For example, you could end up with something like:

        1& NELOO"

Urk -- what a disaster! In a case like this, the easiest
thing to do is type the entire line again.

Move the cursor to the very beginning of the line...and
simply <u>type over the top of the whole thing.</u>

After you've made this sort of correction, it's a very good
idea to LIST your program. Why would you do that? To check
for left-over, incorrect lines that may still be in your
computer's memory!

        If you managed to get the line number right, at
least, there's a rather neater way to correct a
really messy line.
Using the <u>same line number</u>, just type the whole
thing in again. (This time, without the mistakes!)
You see, your computer knows that it can't have two
lines with the same number.

So it'll replace the <u>first</u> line, with the <u>second</u>
(and correct!) version.

        Now, listen closely...'cause we've got a very,
        very important rule to tell you about!
        After making corrections of <u>any</u> sort (whether
        you used RUBOUT, INSERT, or retyped a whole
        line), <u>you must press RETURN !</u> It doesn't
        matter where you are on the line .
    You see, by RETURN ing you are telling your
    VZ-200 to <u>take notice</u> of the corrections you just
   made!

Chapter 5

Now for a little Mathemagic!

Wonderful! Now you've done your first piece of programming.
And it worked! This programming business certainly is a
useful trick, isn't it? But there's another side to your
VZ-200...and it's just as useful.

In a matter of moments, your
VZ-200 can become not a bird...not
a plane...but a

And it doesn't even need to step
into a phone booth! To turn your
computer into a Super Calculator,
it's just a matter of giving
instructions a little differently!
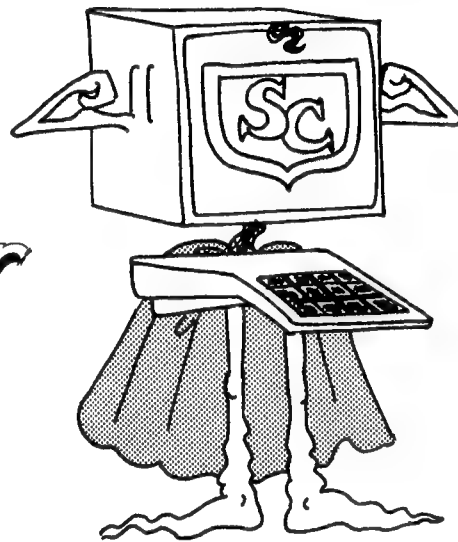
When we wrote our first program, each step had a number.

Beginning each line in your program with a number is like
telling your computer:
"Remember these steps...but don't follow them yet"
The VZ-200 puts your instructions into its MEMORY, so it
won't forget what you've told it.

     Orders like these are called DEFERRED INSTRUCTIONS.

There's another sort of order you can give your VZ-200.
This one says:
"Follow this step...straight away!"

     Orders like these are called DIRECT INSTRUCTIONS.

To get a computer to do maths problems immediately -- like
a pocket calculator -- all we have to do is leave out the
line numbers! Now, we've told you how to turn your VZ-200
into a Super Calculator. Perhaps the next thing to show you
is how to do calculations!

Imagine you wanted to work out a sum all by yourself. If
it's a very easy one, you might use your fingers (and your
toes, if you take your shoes off!). But most of us are used
to writing sums down on a piece of paper when we want to do
them.
And that means that most of us are used to seeing the

24

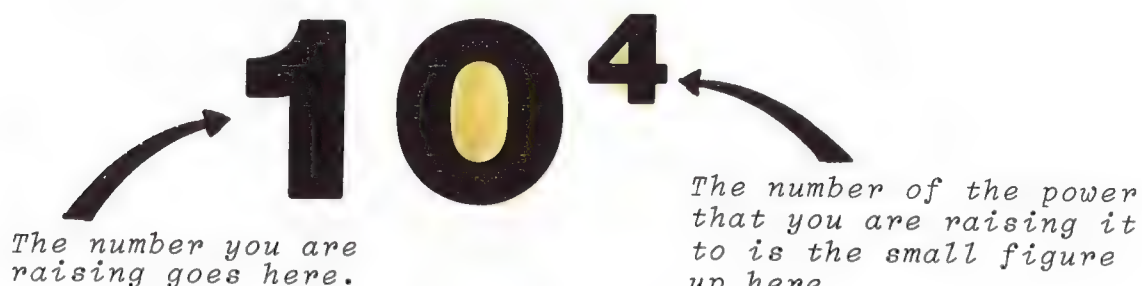symbols for different mathematical functions looking like this:

> addition (or plus) looks like **+**
> substraction (or minus) looks like **–**
> multiplication looks like **x**
> division looks like **÷**

When you're asking your VZ-200 to work out sums for you, adding and subtracting are no problem -- their signs look exactly the same as usual.

But what if you typed the usual sign for multiplication into the computer? Right -- the VZ-200 might mistake it for the letter "x"...and that would only make confuse the poor thing! So when you're talking to your computer, the symbol used for multiplicaton is * (which looks a bit like an "x" anyway!)

The VZ-200's sign for division is different, too--it looks like this: /. If you want your computer to divide say 56 by 8, the sum you'd give it would look like 56/8.

Your VZ-200 can even raise one number to the power of another! Now, most of you know how to write a sum like this on paper. For example, 10 to the power of 4 is written like:

The number you are raising goes here.

The number of the power that you are raising it to is the small figure up here.

> (This is really just a short way of saying 10x10x10x10!)

It would be hard for the VZ-200 to write a sum like that on its screen. So to raise one number to the power of another, we use the **↑** symbol between the two numbers in the calculation.

To get your computer to work out 10 to the power of 4, you'd type in the sum like this: 10↑4

(Just think of it this way--the arrow is there to tell the second number where it's supposed to be!)

We did mention that your VZ-200 can become a Super Calculator...and it really is capable of doing some super complicated sums. But if there are a whole lot of maths operations in one sum, your computer will work things out in a special order:

First: It works out any numbers that are being raised to the power of another (by the way, the very grand mathematical name for this is "expotentiation"). It does this starting from the left, and working its way to the right.

> Eg: to work out 3 ↑ 2 ↑ 2
>
> the VZ-200 first works 3 ↑ 2 (which is 9); then works out 9 ↑ 2 (to get the answer of 81.)

 Second: It does any multiplications and divisions (from left to right).

Third and last: It works out the addition and subtraction (from left to right).

> Eg: to work out 6 + 3*4 + 6/3
>
> the VZ-200 works 3*4 (that's 12), and 6/3 (which is 2). Then it adds 6 + 12 + 2 (to arrive at the answer of 20)

> Incidentally, you might want to use a set of brackets in your problem. If so, whatever is inside those brackets is treated as a separate little sum. It is always worked out before anything else!

> Eg: to work out 18/(3 + 3)
>
> the VZ-200 does the bracketed problem (3 + 3) first--and decides that the answer is six. So the sum it's then left with is 18/6 -- which turns out to be 3.

GETTING VZ-200 TO TELL YOU THE ANSWER

Last chapter, we asked the VZ-200 to write something on its screen. Remember the BASIC word we used to do it? That's right-- the word is PRINT...and we put quotation marks around the thing we wanted to see on the screen .

Lets's try testing it out on a simple sum. Hmm, lets see...how about 2 + 4. First, type it in like this:

        PRINT "2 + 4"   RETURN

Can you see what happened? Your computer did exactly as it was told...it printed what was inside the quotes . Only trouble is, we don't want to see the problem--we want to see the answer!

To get the VZ-200 to work out your problem and
print the answer, simply use the PRINT statement
WITHOUT THE QUOTE-MARKS!

Care to try it out? Use CLS to wipe the screen (it's just
like having a blackboard duster, isn't it?) Now, type in
your little sum again...this time, like this:

PRINT 2 + 4   RETURN

Ta-da! A 6 should now be on your screen -- and that, of
course, is your answer!

Like to add a touch of class to your maths problems? We
can, you know...by using both sorts of PRINT statements.
It's just a matter of putting your sum into a short
program...like the one below, for example:

```
10 PRINT "2 + 4 ="
20 PRINT 2+4
30 END
```

Can you see what it's going to do? Type it into your VZ-200
(don't forget to use the   SPACE   key for spaces and the
RETURN   key at the end of each line, will you!)

Got it? If you don't trust your typing, use LIST just to
make sure that everything is exactly right. (Isn't it great
having all these helpful commands at your fingertips?) If
everything's fine, you can RUN your program.

(Oh, if you'd like to CLS (Clear the Screen) again
first, go ahead! A lot of people like to do this
before they RUN something...it looks a whole lot
neater!)

There! Now, the screen should say

```
READY
RUN
2 + 4 =
6
READY
```

What the program actually said to your computer was this:
    Line 10 said "Print this on the screen, exactly as it
    is."
    Line 20 said "Work out this sum...then print the
    answer."
    Line 30 said "That's all, folks!"

That's a pretty snazzy trick! But do you think there must
be a way of putting the question, and the answer, on the
same line?
Do you think we would have brought it up if there wasn't a
way?

Press LIST again, to put your program back on the screen.
Now, can you remember what we told you about the "editing
arrows"? Use them now, to move the flashing cursor back up

to the end of line 10. (Don't forget, hold the [CTRL] key down all the time that you're moving the cursor around.)

Once the cursor is just where you want it, press the semi-colon (the little symbol that looks like this ";"). Done that? Now press [RETURN] again--the cursor should end up at the beginning of line 20. Using the arrows again, put it back where it belongs--under the very bottom line (READY).

Now, a quick press of the CLS key and you're ready to RUN your jazzed-up program. And when you do, it'll look like this:

```
READY
RUN
2 + 4 = 6
READY
```

Ah, that's even better. Who'd have thought that one little semi-colon could be such an important helper. It told the VZ-200 to stay on the same screen line when it had finished printing line 10!

Getting more adventurous

Whoopee! The more we tell you about this wonderful little computer, the more you'll want to try things of your own. Let's try to get the VZ-200 to solve that same maths problem for us...in a way that's even more elaborate.

Type NEW to make the VZ-200 forget our last program. OK? Now, type in this one:

```
10 PRINT "COMPUTER, ADD 2 AND 4"
20 PRINT
30 PRINT "CERTAINLY, MASTER!"
40 PRINT "THE ANSWER IS";
50 PRINT 2+4
60 END
```

Clear the screen (just to be tidy!), type RUN...and hold onto your hats!
Goodness--would you ever have believed that a computer could be so polite? Look what the VZ-200's saying now!

```
READY
RUN
COMPUTER, ADD 2 AND 4

CERTAINLY, MASTER!
THE ANSWER IS 6
READY
```

Did you notice the little extra trick we sneaked in on you? Where line 20 should have been, there's nothing at all! That's because when you typed line 20 into your program , it looked like this:
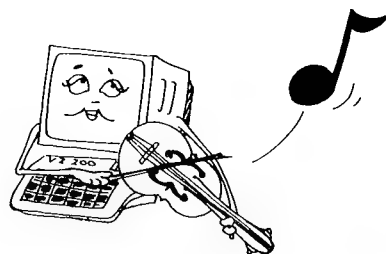   20 PRINT
In other words, you told the computer to print nothing on

that line...and that's just what it did. It printed a blank line.

You can use this trick to make your "print-outs" look neat!

Notes

Chapter 6

Constants & Variables

Sooner or later, every computer programmer will need to know the difference between something that's constant , and something that's variable. We might as well make it sooner...so that difference is what we'll learn next!

WHAT'S A CONSTANT?

If you call something constant, it's just like saying that it never changes--its value is constantly the same. Here's a little example to illustrate the whole idea:

These numbers live on a short number line...from negative 3 to positive 3. Let's imagine that every one of these little fellows has a "personality" (or value) all his own. Each one is different from the others.
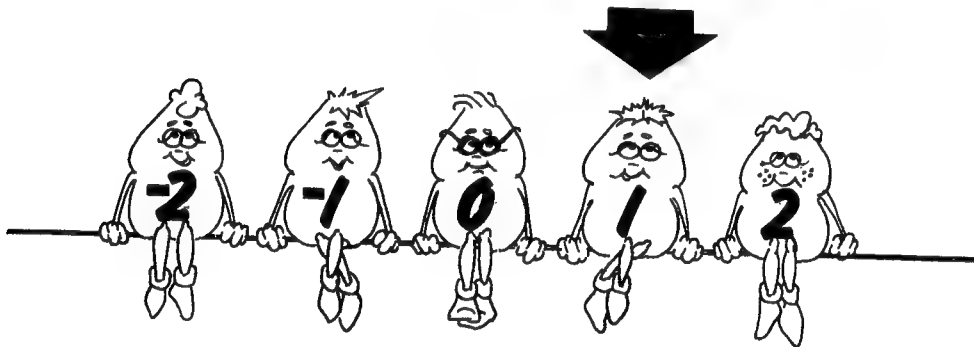
As you'll notice, every number has his own position on the number line. Where they sit depends on their values...and they can never, never leave their spot! Take number 1, for example. He's always got the same value, and he always sits right where he's sitting now. Nothing can change that.

"But hang on!" you might say. "I can change the value of 1...all I have to do is add 1 to it." Well, let's see what happens if we try.

Writing "1+1" is really a short way of saying "Start at number 1, and take 1 step up the number line." When you do that sum, you'll end up at number 2, of course.
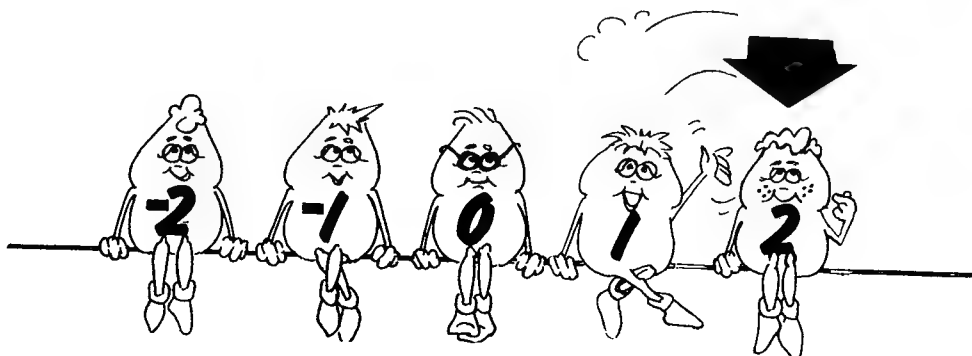
But you haven't changed 1 into 2...you've just
moved along the line to a new number! 1 is still
sitting there--and he hasn't changed a bit!

You can add, subtract, multiply, and divide constants...in
fact you can do whatever you like to them. They won't
change--you'll just end up with different constants as your
answers!

HOW ABOUT VARIABLES?

Have you guessed yet what we're going to tell you about
variables? Right! A variable is something that can change
its value. And it's you who does the changing...as many
times as you like!

Imagine that your VZ-200's
memory is divided up into a
whole lot of compartments...a
bit like the "pigeon-holes"
behind a hotel reception
desk.

Suppose you want the hotel
desk clerk to get a parcel
out of one of these
pigeon-holes. If none of the
compartments are marked with
a name, you can't tell him
which compartment it's
in...and he simply won't know
where to start looking!
Before the desk clerk can
retrieve anything, every
compartment needs to be
labelled with a name of its
own. When you "create" a
variable, you are:
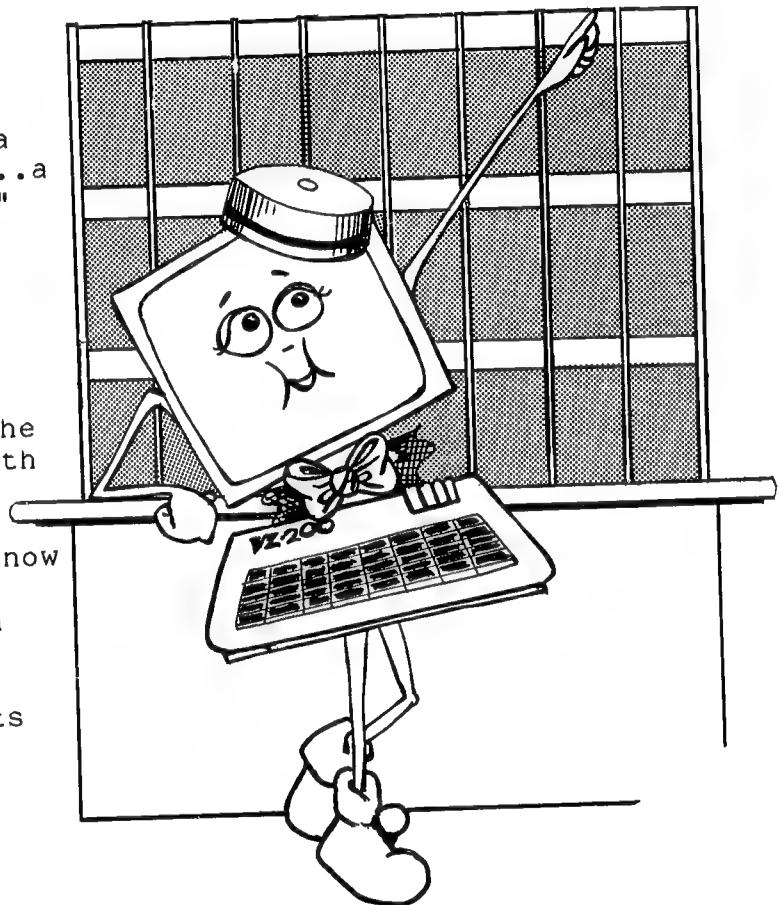
1. Giving a name to one of the pigeon-holes in your
   computer's memory.
2. Putting something into that pigeon-hole!

   A variable name can be:
   •Just plain old A, or B, or C,...or any letter,
   right down to Z.
   • Any combination of two letters--like AB, IQ,
   MT and so on.
   • A letter of the alphabet, followed by any
   number from 0 to 9 -- for example, A3, Y0, R2,
   D2. (If you want to label your compartments
   like this, just remember that the letter always
   comes first! If you try to use a name like 2B,
   or 4U,or 6S, your computer won't like it at
   all!)

The name you give to a variable can be longer than those shown above...but the VZ-200 won't read past the first two characters of a variable name.

> If you label a variable SMILE, the VZ-200 will call it SM. The same rule applies no matter what name you use. For example GEORGE, BANANA and SUPERCALAFRAGALISTIC are just GE, BA, and SU as far as the VZ-200 is concerned!

Warning: the VZ-200 has a special list of "reserved" words. Now, these words -- just like a "reserved" table in a restaurant -- are set aside for some other use. And using one of them as a variable name is definitly a no-no! (If you look in Appendix C at the back of the book, you'll find this list.)

> Unless you've got a whole lot of variables that you want to label, we suggest that you just stick to one-letter names. It's really a whole lot easier!

How to put a variable into the VZ-200's memory

To do this, we'll use a new BASIC command called LET. Look at this example:

LET A=4

The first part of this instruction says: "Label a compartment in your memory as A."

The second part says: "Store the value of 4 in compartment A."
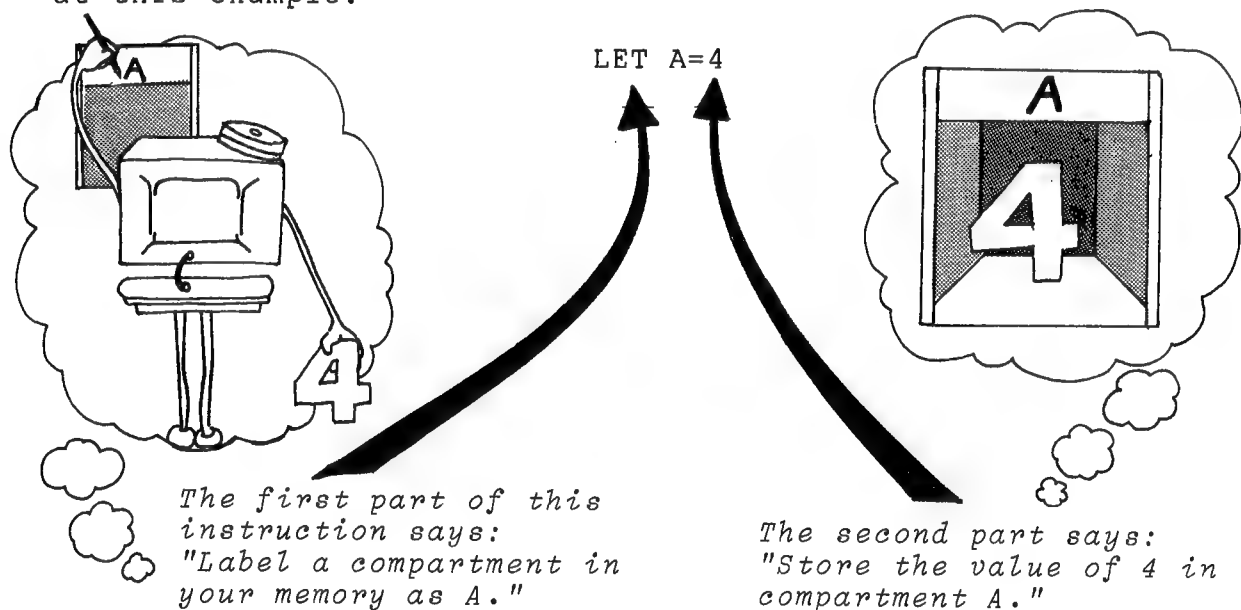
Remember what we told you about variables? You can change their value whenever you want. To change the value of A to, say, 7, simply type:

LET A=7

Your VZ-200 will find the compartment named A in its memory. There's only room for one value in each compartment. So after your computer has "formed" the new value (in this case, 7), it will use it to replace whatever is already inside A.

Easy, isn't it? You can change the value of a variable just by giving it a new one!

33

Now, for some very surprising news! There are a few
shortcuts that a programmer can take...and here's one of
them:
> When you give your VZ-200 an instruction to create a
> variable, the word LET doesn't <u>have</u> to go at the
> beginning! You can leave it out <u>altogether</u> -- your
> computer will understand what you want.

(Really, LET is only there to remind us humans what the
<u>rest</u> of the command is doing!)
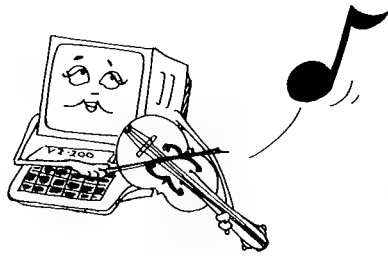
Notes

Chapter 7

More Mathemagic!

Hello? Anyone there? Oh, good...you made it through the last chapter. We're very glad to hear it--because now you know all about the **LET** statement. Not only do we use it to label a compartment in the VZ-200's memory; but also to put a 'value' into that compartment.

That's one way of setting up a variable. This chapter, we'll learn another BASIC statement that'll do a similar job to **LET**. But it goes about doing it in a different way.

The word for this new statement is **INPUT**. Here's the difference:

> LET A=7 says: 1. "Label a compartment in your memory as A."
> 2. "Put a value of 7 into compartment A--immediately."
>
> INPUT A says: 1. "Label a compartment in your memory as A"
> 2. "Don't put anything into compartment A yet...just wait where you are until a value is typed in from the keyboard."

See the difference between the two? If the VZ-200 stumbles upon an INPUT statement when RUNning a program, it'll stop and wait (very patiently, too!) for you to tell it what you'd like the value of A to be.

INPUT really is an extremely helpful word to know. Want to know why? OK...we'll show you!

DANCING ON THE TABLES.
Just about everyone we know absolutely hates having to do "times-tables" (Yuk!). You remember...they look like this:

        2 X 4 = 8
        3 X 4 = 12
        4 X 4 = 16
        ...and so on.

Yes, yes. General groans all round. But never fear! Your trusty VZ-200 has come to the rescue! By the time we've finished this chapter, you'll never have to do a times-table again...'cause we're going to train your computer to do them for you. Ah, what a blissful thought!
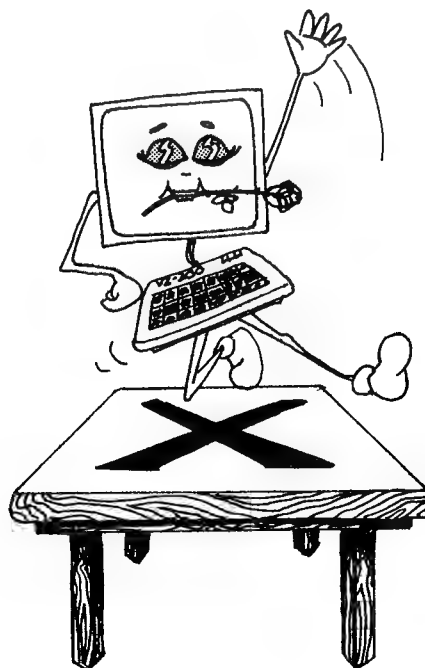
So, let's get started. It's not a difficult
program at all...we'll go through it with you,
step by step.

Before we can do anything, we'll need a number to multiply
by. How 'bout something simple to start with? 4 should do
nicely.

The very first thing we'll need to do in our program is
give the VZ-200 a number to multiply by 4. Wouldn't you
like to have your computer ask you for it, very politely?

We thought so! Perhaps the first line of the program should
look something like this:

```
10 PRINT "WHAT NUMBER, PLEASE"
```

Now, line 20...and now it's time to put our brand new BASIC
word--**INPUT**--into action. Type in the following line:

```
20 INPUT A
```

(Remember, when the VZ-200 is RUNning this program, it'll
stop here and wait for us to give it something to put in
pigeon-hole A.)

What we'll want the next line to do is print our sum, and
it's answer, on the screen. Hey, that's easy! We've done it
all before in chapter 5! Type this into your computer:

```
30 PRINT A;" X 4 =";
```

Hey, just a minute...there are 2 semicolons in that line.
If you noticed them, that's good. We know what the second
one's for--but what about the first? Well, relax. It's just
another little trick to make things easier! You see, in
this line you want the VZ-200 to PRINT:
  ● The number that you put into pigeon-hole A
                    ...and...
  ● The bit in quotes that comes straight after it
By putting a semicolon between these two things, you can
use the same PRINT instruction for both of them...and
they'll be printed side-by-side on the same line.

You probably know what comes next without our having to
tell you! It's the line to get the VZ-200 to do your
calculation and tell you the answer...and it will look like
this:

```
40 PRINT A*4
```

Of course, this is simply saying "multiply whatever is in
pigeon-hole A by 4, and print the answer on the screen."

This is a times-table--and we won't be content with just
one sum! You'll probably want to multiply other numbers by
4, too. So in line 50, why not get the computer to ask for
your next number to go into pigeon-hole A? Here's how we'll
do it:

```
50 PRINT "ANOTHER NUMBER"
```

Right--so exactly _how_ do we get this new number into A. No
sooner said than done, thanks to the next line of our
program. In line 60, we'll use our old friend the GOTO
command. Type this:

```
60 GOTO 20
```

This GOTO command really is a wonder, isn't it? It will
send your VZ-200 zooming back up to line 20--and it'll wait
there while you think up _another_ value to put into
pigeon-hole A.

> Don't forget what happens when you tell
> your computer to store a new value in a
> particular compartment in its memory.
> First, it forms a new value for your
> variable. Then it removes what's _already_
> inside that pigeon-hole -- and puts the new
> value in its place.

And there it is! Your very own program to work out the 4
times-table for you. But why should you take our word that
it works -- when proving it for _yourself_ is as easy as
feeding in RUN ?

Go ahead and try it (and don't forget to CLS first, will
you?)

Hey--what's this? All that's on the screen so far is

```
READY
RUN
WHAT NUMBER, PLEASE
?
```

We did warn you about that! The question-mark in the second
line of your program is just the INPUT statement doing its
job. Your computer wants you to give it a number to
multiply by 4. Well, there's no point in keeping it
waiting! Let's think up a number. Er...what about 2? Type
it in, and press RETURN.

```
READY
RUN
WHAT NUMBER, PLEASE
?  2
  2 X 4 = 8
ANOTHER NUMBER
?
```

Blimey! It does work, after all! See if you can work out
what's happened. The program has "looped" right back to
that INPUT statement on line 20...and now the VZ-200 is
asking for _another_ number to multiply by 4. And it'll keep
on answering your questions, and looping around to start
again, until you're tired of giving it new numbers to play
with. (_Or_ until it's time for your dinner!).

Go ahead! Try a few more numbers, just for the heck of it.

Perhaps you're ready to try another
times-table...the 9 times-table, for example.
Easier done than said! It's just a matter of
changing a couple of lines in your program.

First, press BREAK--it'll stop the VZ-200 in
its tracks.
Next, LIST your program on the screen. Now, use
the editing arrows to change the "X 4 =" in
line 30 to "X 9 =". Then change line 40 so that
it says A*9 instead of A*4.

Don't forget to press the RETURN key after changing each
line!

All done? Move the cursor down to the bottom
again (so that it's flashing underneath READY).
Type CLS, and RUN your new program...you'll
find that the VZ-200 will now multiply any
number you give it by 9!

Now: a little program for the lazy types! Oh, you know who
we mean...those of you who think it's a bit too much
trouble to INPUT a new number each time! Believe it or not,
this program will get your VZ-200 to increase the value of
A every single time it goes round the loop. And do it
automatically, too!

```
10 A=1
20 PRINT A; " X 4 =";
30 PRINT A*4
40 A=A+1
50 GOTO 20
```

Once this program is RUN, you don't have to do a thing!
(Except, of course, to sit back and be amazed.) This is how
it works:

Line 10 stores a value of 1 in a pigeon-hole named A.
Line 20 prints the sum that the computer is about to do on
the screen.
Line 30 actually does the calculation -- multiplying A
(which is 1!) by 4, and printing the answer.
Line 40...ah, here's the secret ingredient in our
"automatic" times-table! This line says "add 1 to whatever
is stored in pigeon-hole A!"
Line 50 "loops" the program back to line 20.

The first time around, your computer will add 1 to
1...giving A the value of 2. Next time, the value
of A will be changed to 3. Then to 4, then to 5,
then to 6...and so on. In fact, the VZ-200 will
just keep on increasing A by 1, and multiplying it
by 4, until you BREAK the program!

That's a pretty nifty trick you've taught your computer!
It'll work no matter what number you want to do a
times-table for -- and you definitely deserve a pat on the
back for mastering it. But whew...this chapter really has
been hard work! You'd probably like a rest now (and a
chance to play around with your new programming skills!).
So we'll leave you alone for a while.
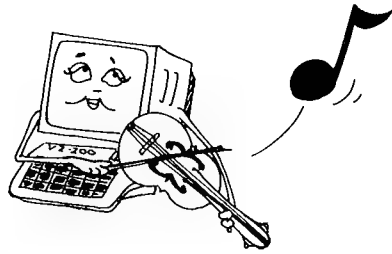
Notes

Chapter 8

Making decisions!

Are you with us so far? Good! Because it's
about time we got started on some really
exciting stuff. We're going to learn about one
of the most powerful BASIC statements you can
ever use in your programs. It's called
IF-THEN-ELSE

As you learn more and more about programming, you'll be
amazed to find just how handy this statement can be. Hey,
now don't go getting all squeamish and scared on us! We
said the IF-THEN-ELSE statement was powerful--but that
doesn't mean that it has to be hard to use. Just wait 'til
you finish reading this chapter. We bet you'll be wondering
what you were worried about.

Putting a program into your VZ-200 is a bit like planning
out a 'jogging track' for your computer to RUN
along...and when you ask it to RUN the program, that's
exactly what it does. It starts at the beginning of the
program (remember? That's the smallest line number), and
keeps on jogging until it reaches the end.

When you put an IF-THEN-ELSE statement into your program,
it forms a sort of 'fork' or 'branch' in the track that
your computer is running along. Now, this is a bit
confusing for the VZ-200. Suddenly, there's more than one
way that it could choose to go...how can it tell which is
the right one? It can't, of course--not all by itself.
You'll have to give it a bit more information, to help it
'decide'!
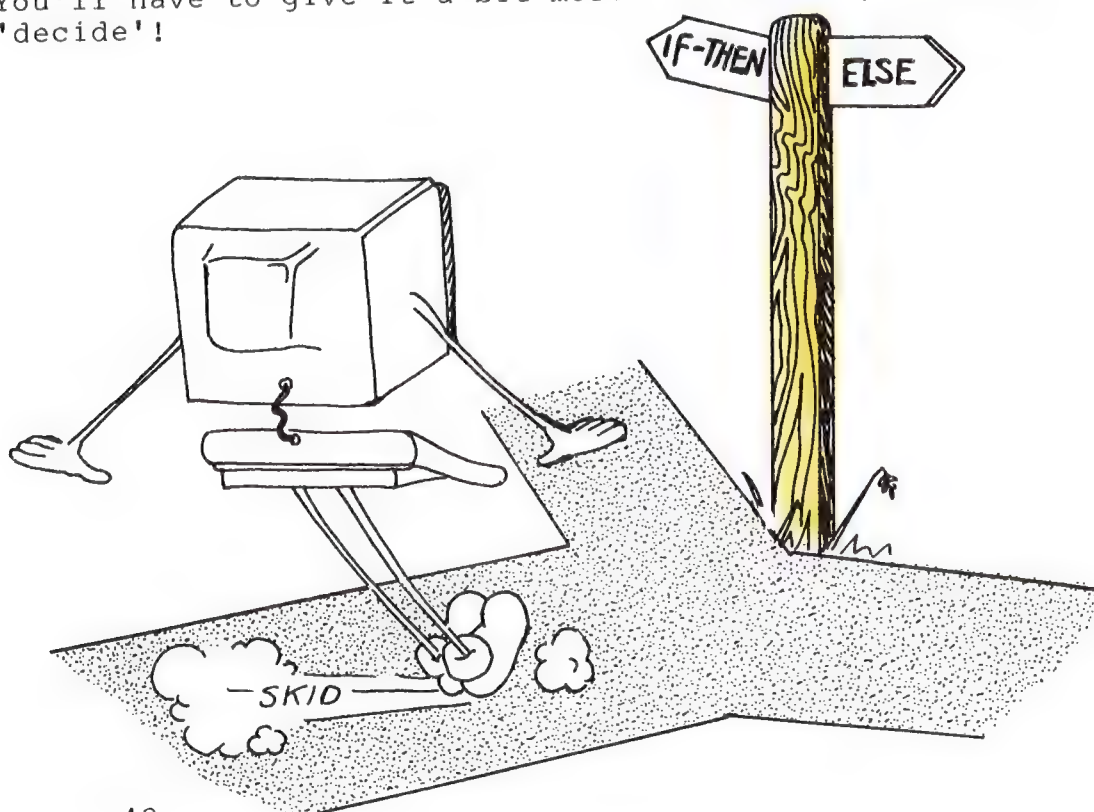
Here's an example that should make the whole idea pretty
simple to understand.

Imagine you are walking down a road in an unfamiliar
suburb, on your way to visit a friend. You want to find
your way to your friend's house...but because you don't
know where you're going, you'll have to rely on the road
signs that you find along the way to give you directions.

So, there you are strolling along. It all seems easy enough
when...Hello! What's this? You've come to a fork in the
road. It branches off to a dirt track in one
direction...and a concrete lane in the other. Which way is
the right way? To decide which path to take, you're going
to need more information. Unfortunately, though, you
haven't got a clue where you are--so where are you going to
get this information from? Never fear! Right at the fork,
there's a road sign to help you. It says:

IF it's not raining, THEN go to the dirt track. Or ELSE, go to the concrete lane

And there's your answer! To
find out the right way to go,
all you have to do is examine
the conditions--in this case,
the weather conditions.

So, how do you go about
examining conditions?
Simple--by testing them!

If you look closely at the message on that sign, you'll see
that it's divided into three parts. See them? One part
starts with 'IF', one with 'THEN', and one with 'ELSE'.
(How convenient!)

>   IF it's not raining
>   THEN GOTO the dirt track
>   ELSE GOTO the concrete lane

Here's how you'll work out the puzzle of which way to go:

>   First part:
>   IF it's not raining...

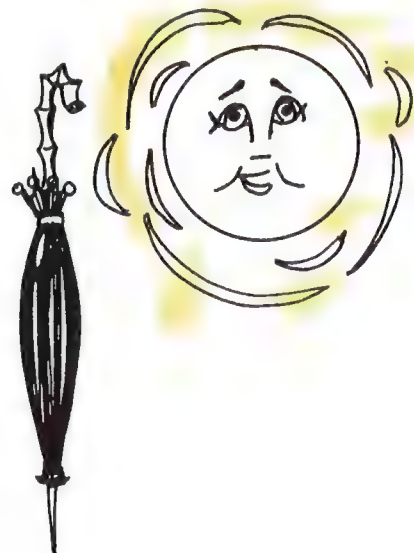>   Ask yourself "Am I getting wet?
>   If you're not, it shouldn't
>   take too long to work out that
>   it's not raining! And if it's
>   not raining, you know that this
>   part of the statement is true.

So you must go straight to the

>   Second part:
>   ...THEN GOTO the dirt track

And do exactly as this part tells you.

But what if you asked yourself "Am I getting wet?" -- and suddenly realised "Yes, I am! Hey, it must be raining." Well, if it is raining, then the first part of the message can't be true. And if it's false, you mustn't follow the instructions in the second part -- you'll have to try something ELSE. Which means you'll have to go to the

Third part:
...ELSE GOTO the concrete lane.

And do as this part tells you, instead!

OK, So now we've worked out how a human being (like you) would deal with a 'fork' in the track that they're following. Actually, we do have a very good reason for telling you all this! We just want to explain that when the VZ-200 comes across a 'fork' in its program, it deals with it in pretty much the same way...with a few differences, of course!

Difference Number 1: If we want to use the **IF-THEN-ELSE** statement to 'force' your computer into making a choice about something, we'll have to give it two things to compare against each other. These things can be either variables or numbers. Computer operators call them 'expressions'.

Difference Number 2: When you were trying to get to your friend's house, the weather conditions were what you were testing. There are lots of other conditions that could influence your decision in other situations...the choice is really quite enormous! The VZ-200, on the other hand, has a much more limited choice of conditions to test. We'll call them "relations"--because what your VZ-200 will actually be testing is how your two expressions "relate" to each other.

This is the list of relations your VZ-200 could use. (To make it easier to write, each relation has a special symbol all its own):

```
 =  Equal to
<>  Not equal to
<=  Less than or equal to
>=  Greater than or equal to
 <  Less than
 >  Greater than
```

Ah, the feeling of power is wonderful. Now you know a way to force your computer into "making up it's mind" about which direction to go--without asking your advice first! As you might have guessed already, this power is going to let you relax a bit...while the VZ-200 takes over more of the work. After all, what else are electronic slaves for?

When making a fork in your program, **IF** and **THEN** are
absolutely essential. ELSE, however, is a different story!
This statement <u>can</u> be left out...if you put the **ELSE**
instructions <u>on the program line directly after the IF-THEN
line</u>! You see, if the **IF** part of the statement turns out to
be false, the VZ-200 will ignore the **THEN** part. Instead,
it'll drop straight down to the next line of the program!
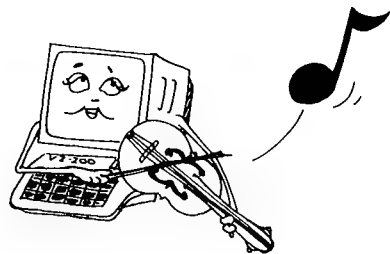(And carry out the instructions there.)

It works like this:

10 IF it's not raining THEN GOTO the dirt track

20 GOTO the concrete lane

Have a bit of a rest now...read back over this chapter,
just to be certain that you understand the **IF-THEN-ELSE**
statement.'Cause next chaper, we'll show you how to put
your new-found power to work!

Notes

Chapter 9

Just playing around...

Good grief! Are we really up to Chapter 9 <u>already</u>?

If you've stuck with us all this way, you must have learned lots and lots about your little computer. In fact, by now you're probably starting to think of your VZ-200 as a friend, rather than a mystery!

We hope so, anyway. Now, what are friends for? For one thing, friends are for having <u>fun</u> with! So this chapter, we'll help you discover how to play games with your VZ-200.

We must admit, the VZ-200 <u>isn't</u> terribly good at a lot of games. For example, you'd <u>never</u> catch a computer playing ice hockey, or ping-pong. Monopoly is a bit too hard for it to manage. And leapfrog is, well, simply out of the question! But there is one sort of game that the VZ-200 does love to play--and that's a guessing game.

So, what are we waiting for? Let's learn how to play!

PICK A NUMBER, ANY NUMBER
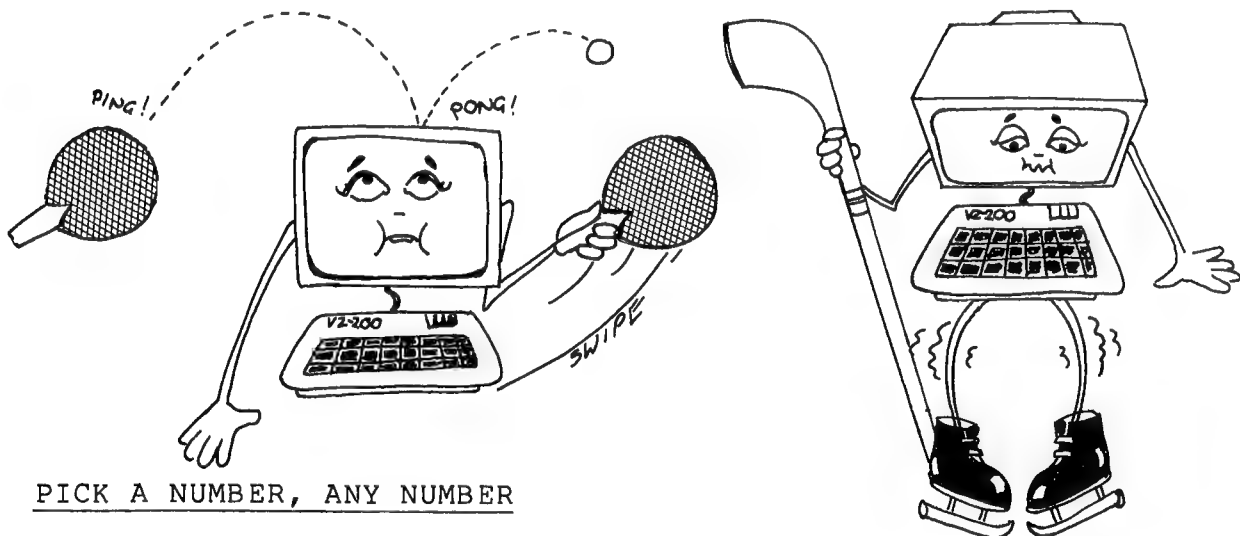
At last, a chance to try out that powerful new statement called **IF-THEN** (no, we don't need to worry about the ELSE this time).

Before we go any further, think of a number--any one you like. Got it? Good--keep it to yourself. This will be our "mystery number"...and nobody will know it except you and your VZ-200. (Don't you just love secrets?!)

And now: on with the game! We're going to write a program that will get one of your <u>human</u> friends to guess your mystery number.

Look closely at the program below. No, don't panic...there's nothing tricky in it! Every command in there is something we've encountered before. You might

actually be able to tell, just by looking, how it's going
to work! (Don't be upset if you can't --we'll go through it
line by line in a moment.) Type each line exactly as we've
shown you.

```
10 A=7
20 PRINT "GUESS MY NUMBER"
30 INPUT B
40 IF B=A THEN GOTO 70
50 PRINT "WRONG! TRY AGAIN"
60 GOTO 30
70 PRINT "YEP, THAT'S IT"
80 END
```

Finished already? You are getting good at using the
keyboard, aren't you! Now, as this is the longest program
we've given you so far, it might be a good idea to LIST
your program, just to make sure that's everything's
perfect. Do that now.

Does it look OK to you? Terrific. Now, here's a handy
little hint. Up 'til now, you've been pressing CLS every
time you'd finished LISTing your program. Wouldn't it be
neater if we put the Clear the Screen command on a line
right at the beginning of your program?

That way, when the VZ-200 RUNs the program, the first thing
it will do is wipe the screen clean. Yes, we thought you'd
like the sound of that!

Underneath what's already listed on the screen, type this
line:

```
5 CLS
```

Now do you see why we leave gaps between line numbers?
Number 5 is smaller than 10...and so our new line will
become the first line in the program. Oh, so you don't
believe us? OK--just press LIST again and see for yourself.

See? There's our CLS line...right at the top of the
program!

Oops...got right off the track for a minute there, didn't
we? Let's get back to explaining how the game works.

ANALYSING THE PROGRAM

Line 10: This is where we whisper our "mystery
number" to the VZ-200, and tell it to hide it away
in a compartment called A.

Line 20: Prints a message on the screen, asking
your friend to try and guess the secret number.

Line 30: Tells the VZ-200 to first label another compartment in its memory as B. It will then wait for your friend to think up a number. When they type in their guess, your computer will put the number into pigeon-hole B.

Line 40: The IF-THEN statement goes into action! There are now two variables in the computer's memory. A contains the mystery number; and B contains your friend's guess. When the VZ-200 reaches line 40, it'll stop and compare whatever is inside A, with whatever is inside B.
Remember, the IF-THEN statement makes a 'fork' in the program...and the VZ-200 has to decide to which way to go. IF B is equal to A, THEN your computer will decide to GOTO line 70--which prints a message that the guess was correct.
But if B is not equal to A, then the IF part of the IF-THEN statement is false...and the VZ-200 will decide to ignore the THEN part and go straight to the next program line (line 50).

Line 50: Prints a message to tell your friend that their guess was wrong...and offers to let them try again.

Line 60: This line loops the program back to line 30, ready for the next guess to be INPUT into compartment B. If they keep on guessing the wrong number, this line will simply keep on looping the program back, ready for their next try. Your friend can have as many guesses as he or she likes!

When they finally hit the jackpot, and the number in compartment B is equal to the number in A, the VZ-200 will GOTO line 70, print the message there, and then drop down to line 80 to end the program--and the game!

Can't wait to try it out on someone? Well, that's alright--we don't expect you to wait. It's all so exciting...you'd better go and find a friend (or two) to play your game with you! We'll be right here when you've finished.

NOW, TO BIGGER AND BETTER THINGS

Congratulations...was your friend impressed?

They certainly should have been...that's a pretty snazzy program you've got there. But gosh, why should we stop now--when by adding just a few more lines, we can make it snazzier still?

The IF-THEN statement really made this program something special. And we only used one of the comparisons (or relations) your computer can use to help choose the way to go! How about adding a couple more, from the list we showed you last chapter?

Oh good--we had a feeling that you'd agree! Press LIST to put your program up on the screen again.

First: change line 50 to say this

```
50 IF B>A THEN GOTO 56
```

(To make the change, all you need to do is move the cursor to the beginning of the existing line 50...and type the new line over the top of it. Don't forget to RETURN at the end!)

When that's done, move the cursor down to the very bottom again and type these lines:

```
52 PRINT "TOO LOW!"
54 GOTO 30
56 PRINT "TOO HIGH!"
```

Yet again, those spare line numbers are coming in handy! Told you they would, didn't we?

Type LIST again. Your computer will sort the line numbers out into their proper order--so your new program should look just like this:

```
5 CLS
10 A=7
20 PRINT "GUESS MY NUMBER"
30 INPUT B
40 IF B=A THEN GOTO 70
50 IF B>A THEN GOTO 56
52 PRINT "TOO LOW!"
54 GOTO 30
56 PRINT "TOO HIGH!"
60 GOTO 30
70 PRINT "YEP! THAT'S IT!"
80 END
```

This program will run in exactly the same way as the last one, until line 50. If the guess in B is equal to the mystery number in A, the VZ-200 will zoom straight to line 70. But if B doesn't equal A, the VZ-200 will drop down to the next program line. And here's where things get more interesting! Since we've changed our program around, we've got a new, improved line 50.
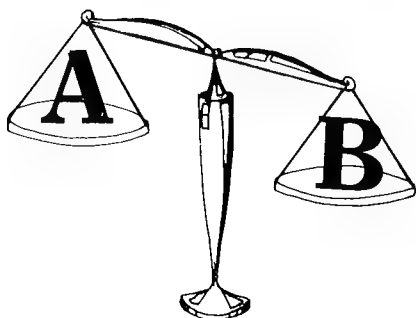
In our jazzed-up program, line 50 contains another IF-THEN fork...and your computer has to make another comparison between B and A before it can decide which way to go. This time, we want to check whether the number in B is larger than the number in A. IF it is, THEN the VZ-200 will GOTO line 56. 56 prints a message to say that the guess was TOO HIGH. It then goes to the line below--line 60. 60 'loops' it back to 30, to wait for your friend to make another guess.

But if B isn't larger than A, the IF
part of the second IF-THEN statement is
false. So the VZ-200 will continue on,
to the next program line in order--line
52.
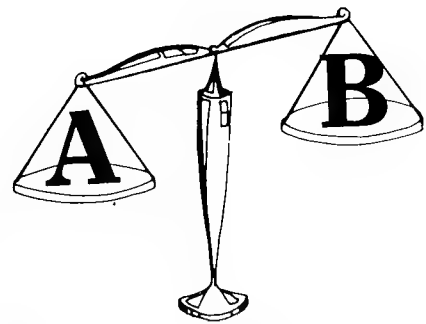
Now, we know that B isn't equal to A. And it isn't larger
than A, either. So if your VZ-200 has finished up in line
52, there's only one thing that B can be--and that's
smaller than A. In this line, the VZ-200 is told to print a
message saying the guess was TOO LOW.

What comes after line 52? Line 54...and this line also
'loops' the VZ-200 back to 30, ready to accept another
guess.

Don't you feel clever? Now you've got a program
for a guessing game that'll actually help the
player...by offering hints along the way!

Who needs a computer that can play leap-frog, anyway?!
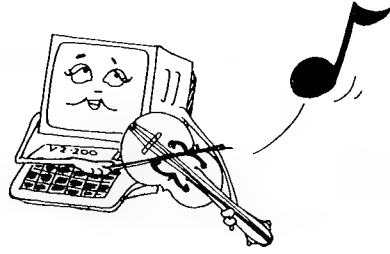
Notes

Chapter 10

Still playing around!

No matter how many friends you've got, there will always be a time when you just can't find anyone to play games with your VZ-200 and you. Well, cheer up...you don't <u>have</u> to wait till your friends are around to have fun with your computer!

You can't really play the guessing game we showed you last chapter, all by yourself. At least, we suppose that you <u>could</u>. But it wouldn't last very long! After all, the "mystery number" is no mystery to you--you're the one who thought it up in the first place. So your very first guess would almost certainly be correct...and the game would be over as soon as it began!

We can soon fix that little problem for you! All we have to do is make a small change to the program at the end of chapter 9.
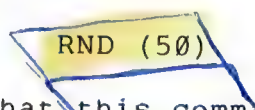
> Once again, it's time to learn another wonderful, helpful BASIC word--**RND**. This command means "Think of a number--but don't tell me what it is!"

We know what you're thinking...if this new trick will get the computer to <u>"think"</u>, it must be magic! Well, it's not exactly that--but it comes pretty close! The **RND** command asks the VZ-200 to pick a number out of the air...almost like a magician pulls rabbits out of a hat.

Now, just a moment. If there's something that there are plenty of, it's numbers. In fact there are millions, upon billions, upon trillions of them (<u>and</u> more besides!). So, what if you let your computer choose any number it wanted? You could spend months sitting there guessing--and <u>still</u> not get it right!

Fair's fair...we'll have to narrow down the range of numbers that your VZ-200 can choose from.

> Directly after the **RND** command, you'll need to put a number in brackets. Just to demonstrate what we mean, we'll use 50. Now, your **RND** command looks like this:

RND (50)

Do you know what this command is telling your VZ-200 now? It's saying "Choose any number you like--<u>as long as it's between 1 and 50!</u>".

In other words, your computer now has 50 numbers in it's hat...and it will pull out <u>one</u> of them to be the secret number.

Ah, that makes things a little
easier! Now you know that the
number you are trying to guess is
somewhere between 1 and 50.
Of course, the number you put
inside those brackets is up to you.
If you want to make the mystery
number easier to guess, you could
use a smaller number...10, for
example. If you'd like to make the
guessing game more of a challenge,
just use a larger number...say,
500 or higher!

Now, let's get back to our
program. If it's still in your
VZ-200's memory from the last
chapter, terrific. All we have to
do is change line 10, so that it
says:

```
10 LET A=RND(50)
```

The new line 10 is saying to your
computer:

> "First, label a
> compartment in your memory
> as A. Now, choose a number
> between 1 and 50, and put
> it into compartment A."

If you turned your VZ-200 off between chapters,
never mind. Just turn back a couple of pages to
find the program, and type the whole thing in
again. Don't forget to substitute the new line
10 for the old one!

And there you have it! A guessing game to play with your
VZ-200,when there's no-one around but the two of you. And
you can play it as often as you like, too...every time you
RUN the program, your electronic friend will think up a
different mystery number for you to guess!

JUST A BIT ABOUT 'FUNCTIONS'

Yep, that word RND sure can make things a lot more
interesting, can't it? Now that we've introduced it you,
we'll tell you what it is! RND is a "function".

What would your answer be, if we asked you what a function
is? Well, if you're not mathematically minded, you might
say that it's an end-of-the-year dance organised by the
social club!
But if you do know a bit about maths, you'll probably reply
(quite correctly) that a function is a sort of 'law'.

There are different laws...and they have different
jobs to do. When we apply the 'law' of a function
to a number (or "value"), the law will carry out
it's own special calculation--and return a new
number value as the answer.

RND is just one of the different functions that you can use
with your VZ-200. Now, a lot of these would be useful for
people who want to solve quite advanced mathematical
problems.

But of course, not everyone is likely to want to do that!
Are you a passionate maths-hater from way back? If so, just
turn the page quickly, or hide under the desk. But
please--not yet! Hear us out for one moment longer.

Like it or not, we're going to tell you about one more
function. Because at some stage, just about everybody (yes,
even you!) will find it useful. It's called SQR, and we use
it to find the square root of a number.

Let's think of a number...16 will do. Here's how we
use SQR to find the square root of our number:

PRINT SQR (16)

When we've typed this line, the VZ-200 will print the
answer on the next line down:

4

Right, now we'll list the rest of the functions
and their uses. If you understand maths--read
through it. If you don't--go and make a cup of
tea!

A LIST OF NUMERIC FUNCTIONS

Function	What it does
ABS (X)	Returns the absolute (positive) value of X
SGN (X)	Returns the sign of the argument X negative returns − 1 X positive returns + 1 X zero returns ∅
SQR (X)	Returns the square root of X. X cannot be negative.
LOG (X)	Gives the natural logarithm of X. The value of the argument must be greater than zero.
EXP (X)	Gives you the value of e^x. e = 2.71828
INT (X)	Gives the greatest integer which is less than or equal to X.

RND (X)	Gives random whole numbers between 1 and X. If X equals zero RND (X) returns random numbers between 0 and 1. X cannot be negative.
SIN (X) **COS (X)** **TAN (X)**	The argument of the trigonometrical functions is in radians. The range of X is $-9999999 \leqslant (X) \leqslant 9999999$.
ATN (X)	This gives the result of ARC TANGENT in RADIANS.
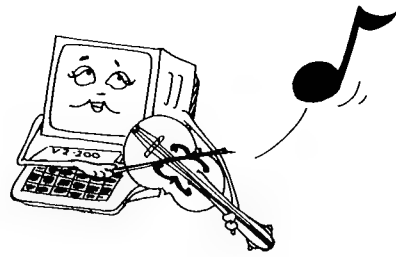
Notes

Chapter 11

Telling your computer "where to get off"!

The more we discover about computer programming, the more exciting it gets! But hey--don't you think it's time we took a bit of a break? After all, everyone has to come up for air sooner or later!

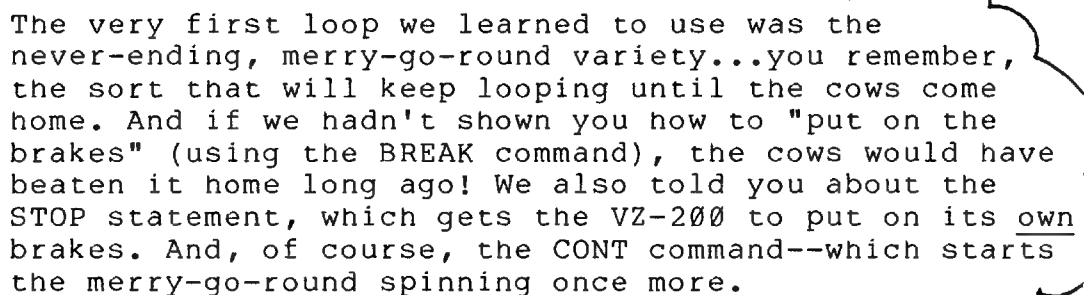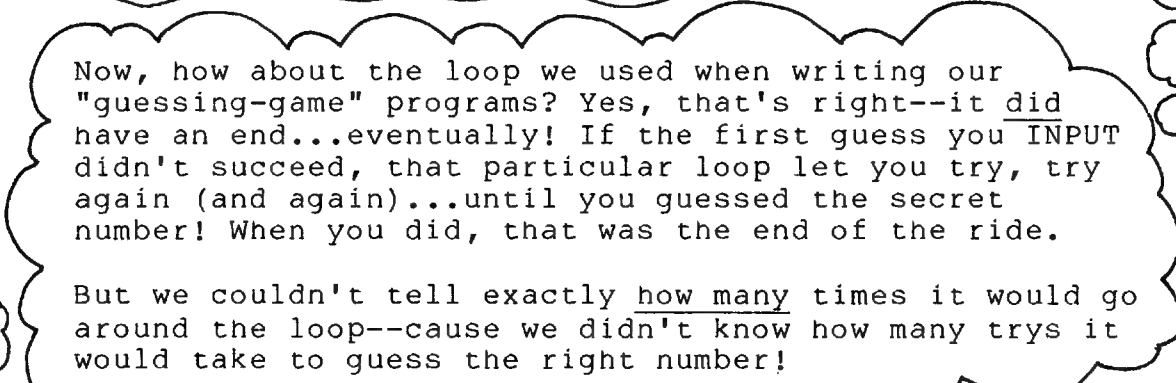That's why this chapter, we thought we'd step back onto familiar ground for a while. So relax and take a deep breath...'cause we're going to learn a bit more about our old friend "looping!"

> "What--again?" you'll complain. "You mean to tell me there's still something to learn about it?"

There sure is! But before we tell you what you don't know, let's think back over what you do.

The very first loop we learned to use was the never-ending, merry-go-round variety...you remember, the sort that will keep looping until the cows come home. And if we hadn't shown you how to "put on the brakes" (using the BREAK command), the cows would have beaten it home long ago! We also told you about the STOP statement, which gets the VZ-200 to put on its own brakes. And, of course, the CONT command--which starts the merry-go-round spinning once more.

Now, how about the loop we used when writing our "guessing-game" programs? Yes, that's right--it did have an end...eventually! If the first guess you INPUT didn't succeed, that particular loop let you try, try again (and again)...until you guessed the secret number! When you did, that was the end of the ride.

But we couldn't tell exactly how many times it would go around the loop--cause we didn't know how many trys it would take to guess the right number!

Wouldn't it be handy if you could set a limit on the number of times the merry-go-round will go merrily 'round? Sort of like giving your VZ-200 a "ticket" for the ride. A ticket that tells your computer how many loops it must make before it's "time to get off"!

Hmm...funny you should mention that! As it happens, there are two ways to make a loop with a fixed end.

LOOPS WITH A LIMIT -- USING IF-THEN

Ah, where would we be without the marvellous statement called IF-THEN? Here's yet another way that it can help us out. As you know, IF-THEN gets your VZ-200 to make a "decision" about something. So why not get your computer to "decide" how many times it has gone around a loop?

We promised to stay on familiar ground this chapter, didn't we? So, to see how this new trick works, let's try it out on something we already know--the little program for working out a 4 times-table (it's the last one in chapter 7).
Type it back into your computer's memory:

```
5 CLS
10 A=1
20 PRINT A; " X 4 =";
30 PRINT A*4
40 A=A+1
50 GOTO 20
```

If we ran that program right now, you know exactly what it would do. (Oh, come on...of course you do!) It will increase the value of A by 1, each time it goes around the loop.

Now, suppose we only want to go up to 12 X 4. That's easy! At the bottom of the program, just type this line:

        45 IF A=13 THEN END

As you know, line 45 will slot into place between lines 40 and 50. Now, our program looks like this:

```
5 CLS
10 A=1
20 PRINT A;" X 4 =";
30 PRINT A*4
40 A=A+1
45 IF A=13 THEN END
50 GOTO 20
```

Want to see it in action? OK...you know what to do. Just tell it to RUN. (And prepare to jump up and down with excitement!)

It works! It works! Thanks to IF-THEN, we've created our very first loop with an end. And how did we manage it? Perhaps you've guessed already.

        By adding line 45, we got the VZ-200 to "test" the
        value of A, each time it changed. IF A was still
        smaller than 13, your computer simply made another
        trip around the loop. But IF A wasn't smaller than
        13 THEN it "decided" to END the loop...and the
        program!

Do you see what this program has done? It's given your computer a <u>ticket</u> for the merry-go-round...one that says "Go around 12 times...and then get off!"

So much for Method 1. It worked beautifully, didn't it? In fact, you're probably <u>so</u> impressed that you just can't wait to hear about Method 2!

Now, now...don't be impatient! Before we can show you the second method, we'll have to teach you three brand new BASIC statements. And that'll have to wait until next chapter...'cause you've done quite enough for the time being!

Notes
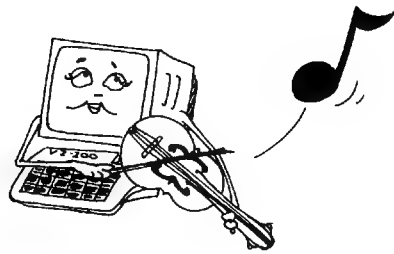
Chapter 12

Still looping (using FOR-TO-NEXT!)

Do you recognize the BASIC statements in the heading of this chapter? Probably not...'cause we haven't told you about them yet!

No, we haven't forgotten what we promised at the end of chapter 11! This chapter, we'll show you that <u>second</u> way of giving your VZ-200 a "ticket" to make it loop <u>a fixed</u> number of times. And the new commands we'll be using? You guessed it--**FOR-TO** and **NEXT!**

FIRST OF ALL: FOR-TO

The <u>first</u> thing that the **FOR** statement says is "mark a pigeon-hole in your memory with the following".
                 (For this exercise, we'll name the pigeon-hole
                 A...but you can choose any letter you like!).
Does that sound familiar to you? Well, it certainly should...the good old LET statement does exactly the same thing! But that's where the similarity ends.

The LET statement gives your computer only one number, or value, to store in that compartment...for example:
         LET A=7.

The FOR statement, on the other hand, shows the VZ-200 a <u>whole group</u> of numbers...say, all the numbers from 1 to 12.

> Now, you <u>might</u> try writing your **FOR** statement like this:
>
> FOR A = 1 2 3 4 5 6 7 8 9 10 11 12
>
> ...but please! Don't! It simply won't work. And even if it did, it really would be a bit tiresome. Especially if you were using a really <u>large</u> number group (imagine trying to write a FOR statement for the numbers from 1 to 100!) No, we've got a better idea. Wouldn't it be easier to use the word **TO**, and write your **FOR** statement like this?:
>
> FOR A=1 TO 12
>
> Ah, yes--that's a whole lot more sensible.

"Sensible or not...that's not going to work!" some of you might say. "<u>All</u> of those numbers won't fit into one compartment, will they?"

If you were one of the smarties who noticed...very good! You're quite right--it wouldn't work. A variable can <u>never</u> have more than one value -- at least, not at the same time.

But don't panic yet! Here's what FOR A=1 TO 12 is really saying:

> "OK, VZ-200--listen closely. I have 12 numbers that I wish to store in compartment A. I want you to change the value of A 12 times...and each time, let A equal the next number in the group."

For example: say the first value of A is to be 1. The next time, A will equal 2. And the next time, A will equal 3. Each number, in turn, is stored in compartment A...right up 'til A=12!

> (Get the picture? It's like asking your computer to walk up a flight of 12 steps--one step at a time!)

Next, comes...NEXT!

Aha! Now for the big question. Exactly how do we get the number in A to change?

We're glad you asked. You see, the **FOR-TO** statement "opens" a loop. So, at the end of the loop, we'll need another sort of statement to "close" it...and send the VZ-200 back to the beginning to start all over again!

Up until now, the statement we've been using to "close" loops is GOTO. As you know, it's a very handy statement. And if we used it at the end of our **FOR-TO** loop, it'd certainly send the computer right back to the beginning. Hang on, though! There's a "but"...and it's a big one! The GOTO statement won't tell the VZ-200 to change the value of A, will it?

Obviously, we're going to need a new sort of statement. And as it happens, **NEXT** fits the bill perfectly!

Here's how we'll use **NEXT** to close the loop around variable A:

        NEXT A

...And here's what it tells your computer:

> "Go back to the FOR-TO at the beginning of the loop...choose the NEXT value for A...and start again!"

LIGHTS...CAMERA...ACTION!

Well, now you know the story of **FOR-TO-NEXT**. Would you like to see them in action? Yes, we thought you would! Let's use a brand new program to put them through their paces.

Hmmmm...anybody got any suggestions for a job to give your computer?

Eureka! (We don't have a clue what that word means...but we do have a bright idea!) Why don't we get the VZ-200 to work out the 4 times-table up to 4 X 12 again--just as we learned to do last chapter? Only this time, we'll use the FOR-TO-NEXT statement instead of **IF-THEN.**

All agreed? OK--here we go. Type in this program:

```
 5 CLS
10 FOR A=1 TO 12
20 PRINT A; " X 4 =";
30 PRINT A*4
40 NEXT A
50 END
```

Now, for the big moment! (Drum-roll, please!). RUN the program...and watch this!

Ta-da! If you did everything right, your VZ-200 should have done the 4 times-table up to 4 X 12...just as it did when we used **IF-THEN!**

Feeling pretty impressed? If so, stay tuned for the next exciting episode (well...the next exciting chapter, anyway!) Because there's even more to learn about **FOR-TO-NEXT.**

Notes
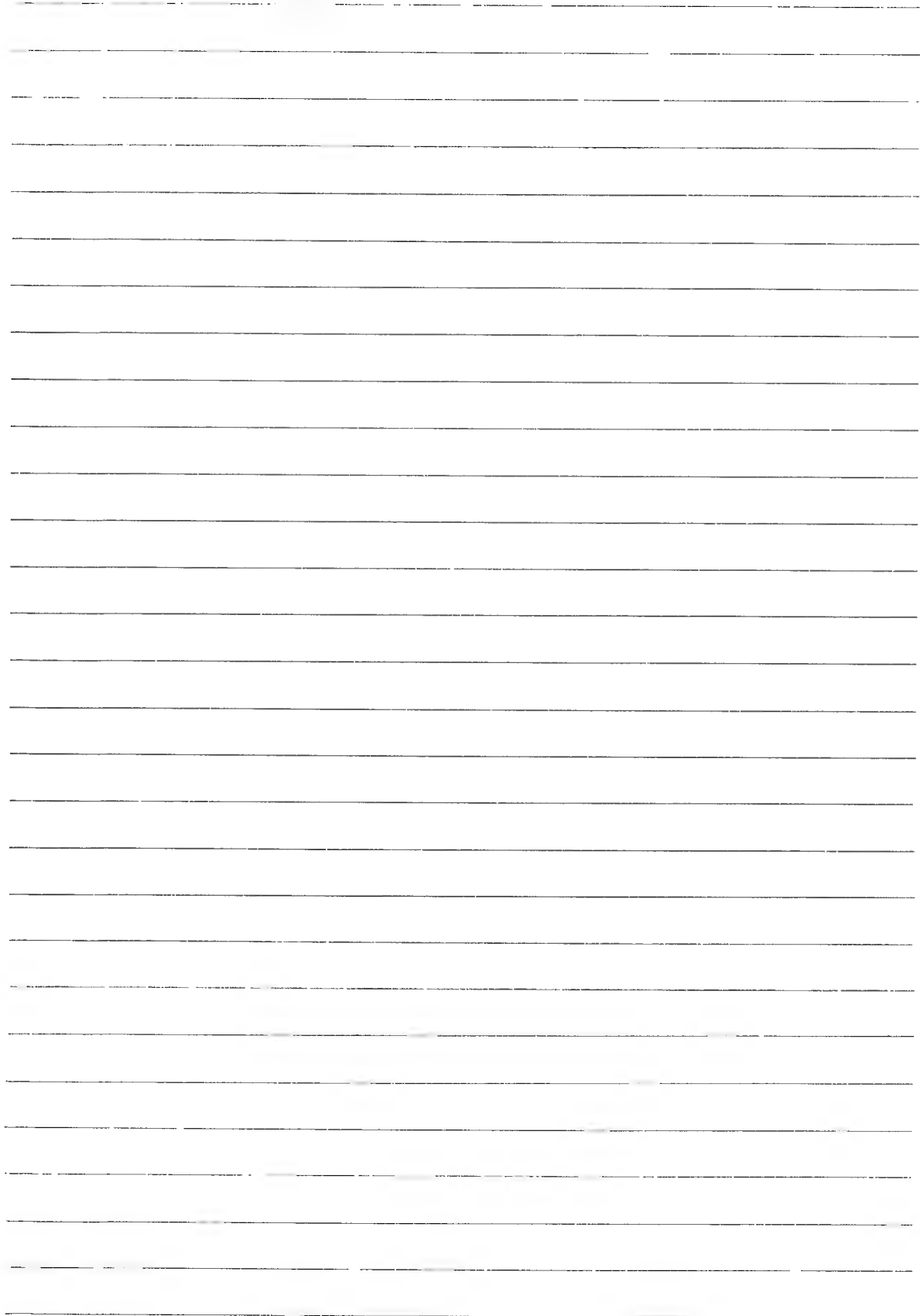
Chapter 13

Even more about FOR-TO-NEXT!

When we first raised the subject of **FOR-TO-NEXT**, we'll bet you never dreamed that it would turn out to be so powerful!

And now, something else to amaze you! By tagging one more BASIC word on the end, we can get **FOR-TO-NEXT** to do even <u>fancier</u> tricks!

Do you ever run up a flight of stairs <u>two steps at a time?</u> Well, our new word--called **STEP**--can ask the VZ-200 to do the same sort of thing. And do it even better than any human could! Using **STEP,** you can order your computer to climb the stairs three at a time! Or ten at a time! Even <u>100</u> at a time, if you like!

Here's an example of how it is used:

        FOR A=1 TO 12 STEP 2

And here's what it makes your computer do:
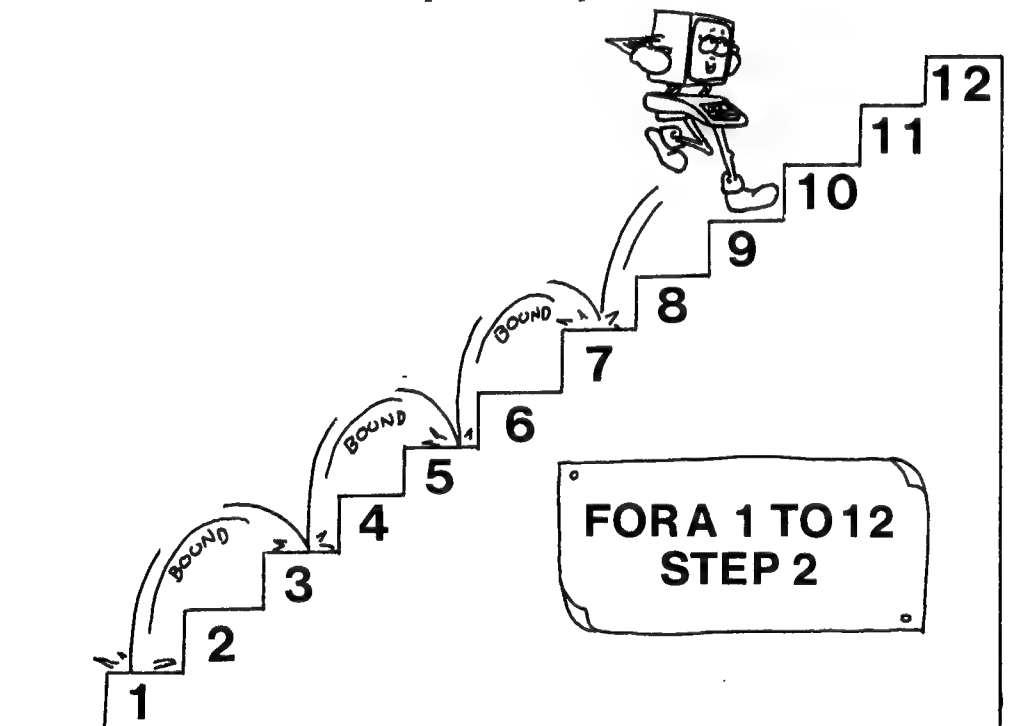
Another talent of the **STEP** instruction: we can use it to actually write a "backwards" **FOR-TO-NEXT** loop. That is, one that uses the highest number first--and works its way down to the smallest!
Amazing! Here's an example:

        FOR A=12 to 1 STEP -1

Here's what it makes your computer do:
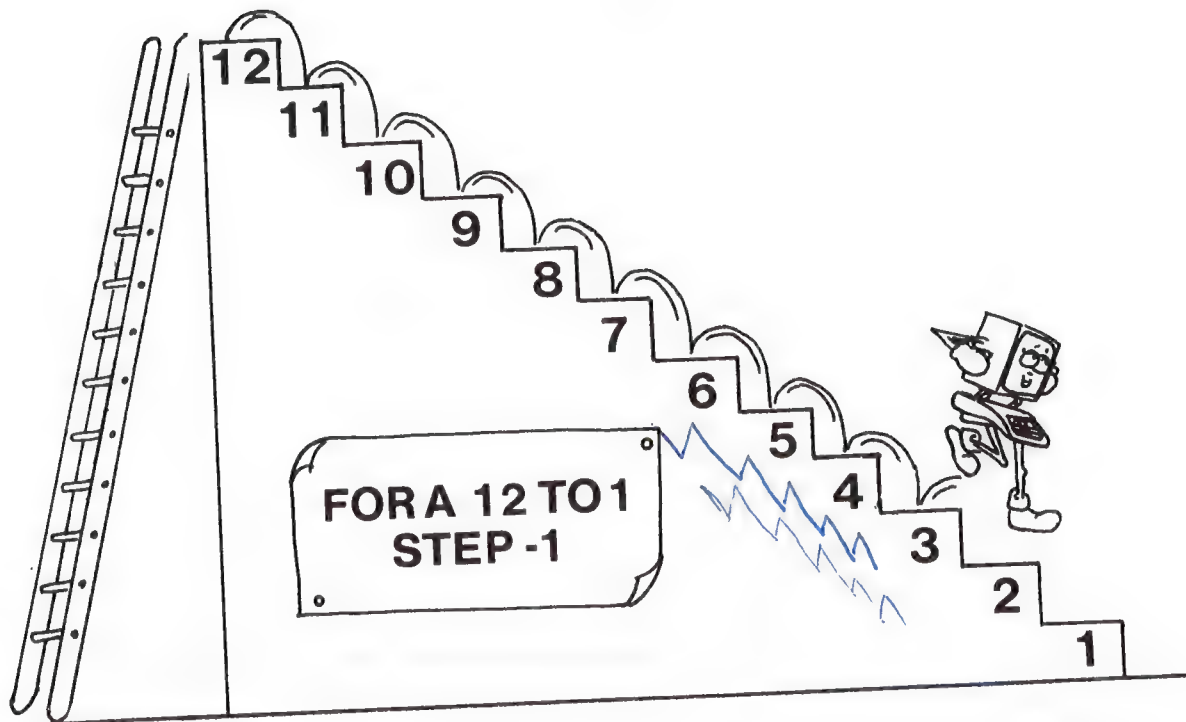
FOR A 12 TO 1 STEP -1

A loop that goes in <u>reverse</u>? This we've got to see! Type in
this program:

```
5 CLS
10 FOR A=10 TO 1 STEP -1
20 PRINT A; " GREEN BOTTLES,"
30 PRINT "HANGING ON THE WALL,"
40 PRINT A; " GREEN BOTTLES,"
50 PRINT "HANGING ON THE WALL,"
60 PRINT "AND IF 1 GREEN BOTTLE"
70 PRINT "SHOULD ACCIDENTALLY FALL,"
80 PRINT "THERE'D BE "; A-1 ;" GREEN BOTTLES"
90 PRINT "HANGING ON THE WALL!"
100 NEXT A
110 END
```

Now, just about everyone knows how <u>this</u> song goes! It
starts out with 10 green bottles...and each verse knocks
one bottle off the wall. (It's <u>supposed</u> to be
accidental--but we're not so sure about that!) Every verse
starts out with <u>1 bottle less</u> than the previous one...and
when all ten bottles have fallen, that's the end of the
song.

<u>A closer look.</u>

We already know what the **FOR-TO** statement in line 1 will do!
The first value to go into pigeon-hole A will be 10, the
next will be 9...and so on, until A=1.

The <u>next</u> 4 program lines print the current value of A,
followed two lines of the song. (This is done twice).

Lines 60 and 70 tell the watcher that 1 green bottle is
about to fall.

Line 80 has several jobs. It:

- Calculates how many bottles will be left if 1 is subtracted from A; and then...
- Prints the answer to the sum in the middle of a song line.

Line 100? This "closes" the loop, and sends the program back to the **FOR-TO** statement at the beginning -- where it stores the next value in A.

When the computer has looped around 10 times (and therefore printed all 10 verses of the song!), it will drop down to line 110 -- and end it all! (No, don't worry! We mean that it'll end the program, not commit suicide!)

Well, what are we waiting for? Let's RUN it -- and see a reverse **FOR-TO-NEXT** loop in action!

Oh...that was a bit disappointing! It certainly works. But hey...don't you think that it works a bit too well? With all those lines whizzing up the screen at such a speed, it's really a bit hard to read the words of the song.

Unfortunately, that's one of the problems with computers. They do just as they're told...but they sometimes do it so darn fast, we poor humans simply can't keep up!

To slow this program down, we'll have to get the VZ-200 to pause occasionally! And to "buy" that extra time, we can give the VZ-200 something to keep itself busy...while we're reading the screen!

There is a way! Look at the line below:

```
FOR T=1 TO 3000:NEXT T
```

We know that a **FOR-TO** statement opens a loop--and a **NEXT** statement closes it again. So: the loop above is an "empty" one ...because we've no sooner opened it, than we close it again!)

It doesn't perform any particular function--it merely sends the VZ-200 "running up some steps". A whole 3000 of 'em! Because your computer is so speedy, we need a staircase of this size to keep it busy for long enough !)

Now, where should we put this "empty loop"? Well, the most logical place would be after the last line in each verse. So LIST your program, and let's see if we can find that spot.

There it is! Line 90 prints "HANGING ON THE WALL"...and that's the very end of the verse. So we'll add our empty loop -- the one mentioned above -- on another line (say, number 95). This'll make the VZ-200 wait a while, before looping back to print the next verse.

OK? Try RUNning it again, to see if we've helped matters.

Ah, that's much better! At the end of each verse--while the VZ-200 is madly dashing up that flight of 3000 stairs going

nowhere--you get a chance to read the song at leisure!

Very appropriately, a loop that "delays" the VZ-200 (or "stalls for time") like this is called...a _delay loop_.

Now... a word about "nested FOR-TO-NEXT loops".

Do you think that sounds terribly technical and confusing? Boy, have we got a surprise for you. We've _already used one!_

Truly, we have...and only a moment ago too! Our little "time delay" loop just happens to be a nested loop as well, (So there you are--they can't be _too_ difficult. Not if you managed to use one without even _realising_!)

If we say that something is "nested", we mean that it is _completely inside_ something else.

So, have you twigged what we're about to tell you? Right--a "nested" FOR-TO-THEN loop is one that's _completely_ inside another FOR-TO-NEXT loop!

If we tried to draw one, it would probably look a bit like this:
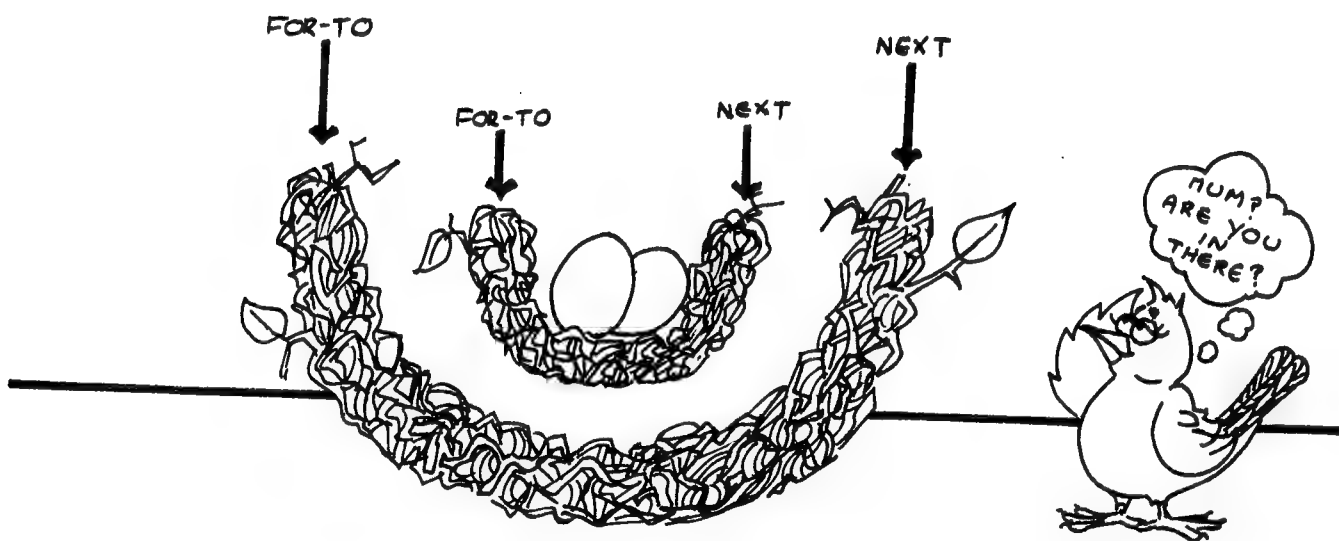
(Incidentally--a nested FOR-TO-NEXT loop doesn't necessarily have to be empty. It can have absolutely anything...as long as the _entire loop_ is inside another loop!)

> _Mystery question_: Did you notice something unusual about our new line, number 95? No? Well, here it is again...take a closer look.
>
>   FOR T=1 TO 3000 : NEXT T

Mystery answer: Line 95 contains two BASIC
statements! The first is FOR-TO...and the
second is NEXT!
Usually, each of those statements would have
had to go on a line of their own. But yes, you
are allowed to pull a stunt like this one....
as long as the statements are separated by a
colon!

Notes
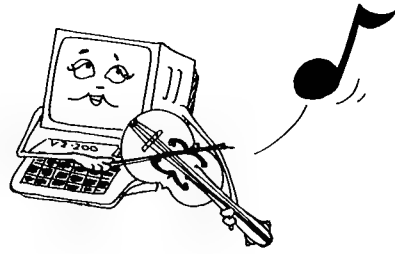
Chapter 14

Programs Within Programs!

As clever as your VZ-200 is, there is a limit to it's wonderful capabilities. (Shock! Horror!) Yes, truly...there is a _limit_ to the number of program lines that it can remember at one time. Each line, you see, takes a certain amount of space in your computer's memory.

And if there's one thing a computer programmer just _hates_ to do, it's waste space! (Especially when that space might be needed for a very, very long program.) Now that _you're_ well on the way to becoming a fully-fledged computer programmer, you probably couldn't agree more.

Do you know the best way of wasting space? By repeating yourself unnecessarily! And occasionally, when you're typing a program into your VZ-200, you'll notice a bit of repetition sneaking in.

To find an example, let's turn back to the "10 Green Bottles" program in chapter 13. What's that you say? You thought it was pretty nifty? So did we! And yes--it _is_ a clever program. But it's not quite perfect...as you'll see if you look closely at lines 20, 30, 40 and 50.

Gasp! 20 and 30 are doing _exactly_ the same job as 40 and 50! Now, that's definitely a good example of line-wasting!

("Aha!" you'll think. "I wonder if they're leading up to something?")

As a matter of fact, yes--we are! This chapter, we're going to teach you how to _avoid_ this sort of repetition...using two new BASIC commands called **GOSUB** and **RETURN.**

Close your eyes...think very hard...and try to remember way back to Chapter 3. Can you recall how we described a program to you? That's right--we said that a program is a _set of steps_, designed to do a special job.

Now, here's a very interesting thought. Lines 20 and 30 are actually a _set of steps_, too...even though the set is only two lines long.

Can you see the special job that this pair of steps does? It tells the VZ-200 to print two lines of the "10 Green Bottles" song on the screen. (And lines 40 and 50, because they contain exactly the _same_ instructions, are really just lines 20 and 30 typed all over again!)

A "mini" set of steps? That's really a sort of "mini" program!

Hmm...it's a bit wasteful to type the same set of steps twice. Are you thinking what we're thinking? Maybe we could have just one set...or mini-program...and tuck it out of the way until it was needed.

Yes, that sounds like a terrific idea! But there must be a name for a program within a program...what do you think it could be? A sub-program, perhaps? Well, that's pretty close! Actually, it's called a "subroutine".

A subroutine is still a part of the main program--but it's kept separate from the rest. You can ask your VZ-200 to use a subroutine at various points of the main program, whenever it is needed. To do this, we simply need to direct the the VZ-200 to:

●Jump to the subroutine, and follow the steps it contains
        ...and then...
●Jump straight back to the next line of the main program again!

Oops...we nearly forgot to tell you something, didn't we! You're probably wondering just how to separate a subroutine from the main program!

The easiest way to keep any two things from getting mixed up together, is to put a "barrier" between them. Agreed? So, to keep your subroutine right out of the way, just give each of its lines a much higher line number than the last line of the main program. (And that line, of course, is the one with the END statement!)

Try starting your mini-program all the way down on line 3000. There's really no danger of it getting mixed up with the main program...because the VZ-200 will always END the program before it reaches it!

In other words, the END statement on the main program's final line acts as a barrier (or "safety fence") to keep the subroutine out of harm's way.

SETTING UP A SUBROUTINE, USING GOSUB AND RETURN.

Getting your computer to jump to a subroutine, and back again, is no trouble...you only have to ask. There's only one catch -- you have to ask it right! And this is where we get use our two new BASIC words -- GOSUB and RETURN.

"GOSUB...is that something like GOTO?"
As a matter of fact, yes! GOSUB is really a special sort of GOTO command -- telling your computer to go to the beginning of a subroutine. And because it's no use asking the VZ-200 to go somewhere without telling it where, GOSUB is always followed by a line number.

To send your computer to a subroutine beginning
on line 3000, you'd type:
                    GOSUB 3000

Maybe you're wondering why we bother with the **GOSUB**
command at all...when GOTO 3000 would do the job
just as well!
Ah...in the world of programming, there's a reason
for everything. GOTO actually won't work in a case like
this. You see, the GOTO statement doesn't warn the VZ-200
that it's being sent to a subroutine! It merely says: "Jump
to line 3000--immediately!"

> Being such a co-operative little fellow, your
> computer will do exactly that. Without stopping to
> notice where it's jumping from! (Oh dear--that'll
> cause a bit of bother. How's the VZ-200 going to
> find it's way home again?)

> Looks like we do need **GOSUB** after all. This
> command is a warning for your VZ-200. It says:
> "Look before you leap! I'm sending you to a
> subroutine now...make sure you can find your way
> back here."

Right...now we know how to send your computer to that
separate little program. And what happens when it reaches
the end? Why, everyone knows it's time to jump back to the
main program. (Don't they?)

Well, er, no -- not quite everyone! You know...and we
know...but don't forget about your computer. As we've
mentioned, it doesn't know anything unless you tell it!
That's why you must always end your subroutine with the
RETURN command. This is just like saying:
> "OK, that's the end of this subroutine. RETURN to
> the main program -- to the line after the GOSUB
> command that sent you here."

OK! Now that we understand the basic idea of subroutines
(and how to use them), let's try out our new skills by
improving the "10 Green Bottles" program!

A subroutine to do the job of lines 20 and 30 (and lines 40
and 50) would look like this:
```
3000 PRINT A;" GREEN BOTTLES"
3010 PRINT "HANGING ON THE WALL,"
3020 RETURN
```

And if we use it to update our little song program, the
new version will look like this:

```
5 CLS
10 FOR A=10 TO 1 STEP -1
20 GOSUB 3000
30 GOSUB 3000
40 PRINT "AND IF 1 GREEN BOTTLE"
50 PRINT "SHOULD ACCIDENTALLY FALL,"
60 PRINT "THERE'S BE "; A-1 ;" GREEN BOTTLES"
70 PRINT "HANGING ON THE WALL!"
100 NEXT A
110 END
3000 PRINT A;" GREEN BOTTLES,"
3010 PRINT "HANGING ON THE WALL,"
3020 RETURN
```

See how it's going to work? Just to make sure that we've
put you in the picture, we'd like to show you...a picture!
Take a look at the diagrams below. (It should make
everything well, pretty clear!)

Lines 20, 30, 40 and 50 of the old program have
been replaced by just two lines -- a new line
20, and a new line 30. Each of these
replacement lines is much shorter now,
too...each one simply says GOSUB 3000!

When you tell your VZ-200 to
RUN the new program, it'll
start off in exactly the same
way. Until it runs into the
new line 20!

Line 20 sends the VZ-200 to subroutine 3000. Your computer will zoom around the subroutine...and RETURN to the next line of the main program. (That is, line 30.)

GOSUB 3000
GOSUB 3000
LINE 40
LINE 30
LINE 20
main program
END
RETURN
3000
ZOOM

Line 30 just happens to contain another GOSUB 3000 command! So the VZ-200 will go straight down to the subroutine again!

This time, it'll RETURN to line 40, ready to continue with the rest of the program

GOSUB 3000
GOSUB 3000
LINE 40
LINE 30
LINE 20
main program
END
RETURN
3000
ZOOM

That's it, folks! Subroutines are very easy to use, aren't they? And -- as we've just discovered -- they're also very good at preventing space-wasting in programs. (In fact, whenever the same set of steps crops up over and over again, a subroutine is worth it's weight in gold!)

Even better:

What if you find several different sets of steps that repeat themselves in a program? Why, just make _more_ subroutines to do _their_ jobs, too! There's no limit _to the_ number of subroutines you can use. (Just be sure to give each one a different set of line numbers. For instance, start one routine on line 3000, one on line 4000, and one on line 5000!)

Best of all:

Remember what we told you about "nested" FOR-TO-NEXT loops? If you do, that's good! 'Cause you can use the very _same_ idea to make "_nested subroutines_"!

EXAMPLE:

You can ask your computer to go from the main program track to subroutine 3000. And subroutine 3000 can contain another GOSUB command...sending the VZ-200 to subroutine 4000. Yet another GOSUB command in subroutine 4000 will send it to subroutine 5000!

The same GOSUB-RETURN rules apply no matter how many subroutines you "nest" together:
A RETURN command will send the VZ-200 back to the line following the GOSUB command that sent it to the subroutine!
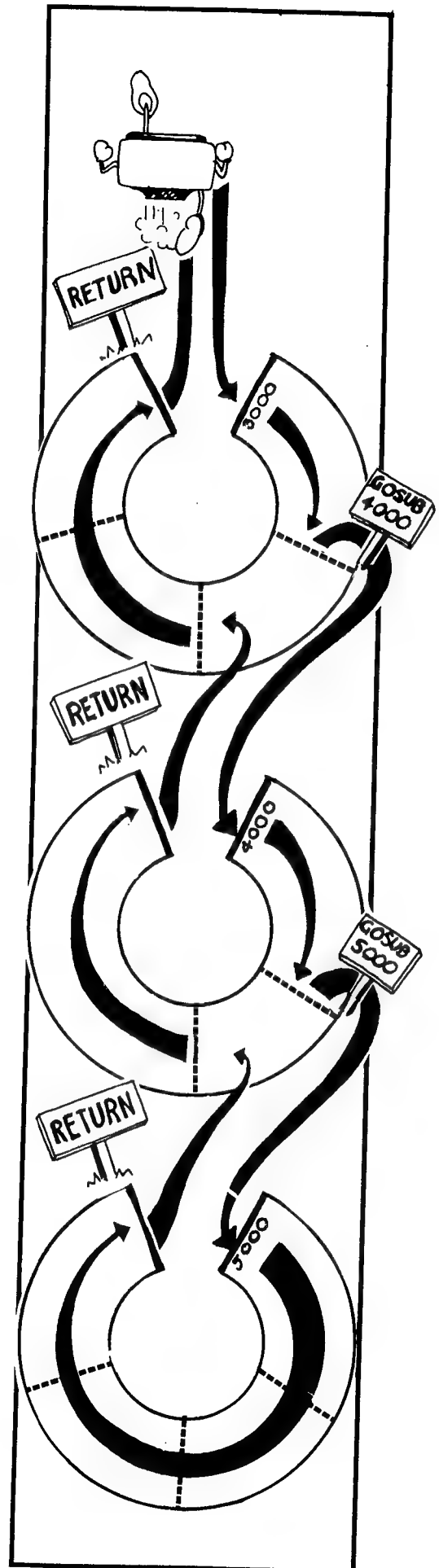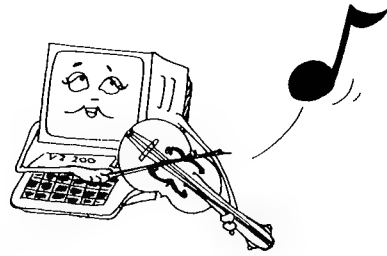
Notes

Chapter 15

Just Stringing You Along...

Boy -- you're really going to <u>love</u> this chapter. And you're
going to love what it can do for your programs, too!
Because right now, we're going to learn a brand new -- and
very exciting -- type of variable.

We already know that the "pigeon-holes" in the VZ-200's
memory are made especially for storing things. In fact,
we've already tried storing <u>numbers</u> inside them -- with
great success. But gosh...it seems such a pity to use them
for only one purpose.

Have you ever wondered if these compartments could be used
to store <u>other</u> things? Like words...or even phrases? It'd
certainly be handy if they could.

Aha! Now here's some good news: Yes, they can!

It does sound pretty amazing...but it definitely is
possible. Why, there's even a special name for variables of
this type! They're called <u>strings</u>.

That seems like an odd title -- at first. But when you
<u>really</u> think about it, the name is quite a sensible one!
You see, forming one of these variables is rather like
making a <u>string</u> of beads! And that's easy to to...it's just
a matter of threading a group of beads together.

> A string is formed in exactly the same
> way -- simply by "stringing" a whole
> bunch of letters, spaces and other
> characters! Now, if you do that you
> will have formed a string constant.
>
> To form a <u>string variable</u>, we just need
> to mark a pigeon-hole with a name of
> our choice...ready to be have a string
> of characters stored inside it!
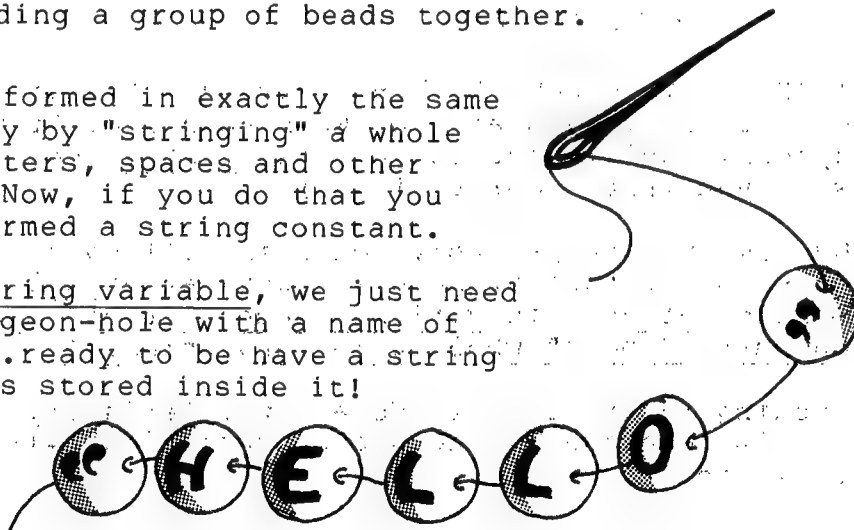
Rules and regulations
Like most new things we've learned, there <u>are</u> a few rules
that you'll need to know. Don't worry, though -- they're
not hard to remember!

> ● A string mustn't be any longer that 255
> characters. (And yes, that <u>does</u> include spaces.
> Why, you ask? Well, those pigeon-holes
> are only so big, you know. If you try to cram a
> huge bundle of letters into one of them, it

simply won't fit!

● The name that you give to a string variable
must always end with a dollar sign! (No, this
doesn't mean that strings are very expensive
things! It simply warns your computer that the
pigeon-hole contains a string of characters,
not a number.)

For example: A$, B$, C$ and D$ are possible
names for strings.

● When you are typing a string into your
VZ-200, it must always be "enclosed" in
quotation marks! For example:
           A$="EENIE"
           B$="MEENIE"
           C$="MINIE"
           D$="MO"

You can treat these new variables in pretty much the same
way as ordinary variables. Oh -- with one exception!

        You can ONLY do 1 maths operation with string
        variables -- and that's ADDITION!

To add strings together, simply give your VZ-200 an
instruction like this one...

        PRINT A$+B$+C$+D$

...and when RUN, the result will look like this:

        EENIEMEENIEMINIEMO

(Oops! Notice how the VZ-200 ran all the strings together,
without leaving spaces between each one? If you want
spaces, you'll have to put them inside the quotation marks
when you write your strings.)

By now, you're probably imagining all sorts of wonderful
ways to use this brand new trick! After all -- at the mere
mention of certain variable names, it can make your
computer "speak" whole words and phrases!

Itching to try it out? So are we! Let's try and get the
VZ-200 to actually "hold a conversation" with anyone who
talks to it!

Enter this program:

```
5 CLS
10 PRINT "HELLO! I'M YOUR VZ-200."
20 PRINT "WHAT'S YOUR NAME";
30 INPUT N$
40 PRINT "HELLO, ";N$
50 PRINT "I'M VERY PLEASED TO MEET YOU."
60 PRINT "WHAT SUBURB DO YOU"
70 PRINT "LIVE IN";
80 INPUT S$
90 PRINT "DO YOU LIKE LIVING IN"
100 PRINT S$;
110 INPUT A$
120 IF A$="NO" THEN GOTO 150
130 PRINT "GOOD! THERE'S NO PLACE LIKE HOME!"
140 END
150 PRINT "OH DEAR! MAYBE YOU SHOULD MOVE!"
160 END
```

Now, if you go through this program very slowly, you should be able to work out what it does all by yourself!

(Oh, don't worry. We know you can do it!)

A few hints: Line 20 asks your friend to INPUT their name. Line 30 "listens" to their answer...and stores the information in a pigeon-hole labelled N$. Lines 50, 60 and 70 do the same sort of thing -- this time, asking for your friend's suburb and storing it in pigeon-hole S$.

Once that information is safely tucked away in the VZ-200's memory, it can use it to make the "conversation" amazingly personal. (You might even fool your friend into believing that your computer really does have a mind of its own!)

A Word of Warning

It's all very well to impress your friends with your computer's courtesy. But be careful! Let your VZ-200 be too charming, and you could have trouble getting it away from them!
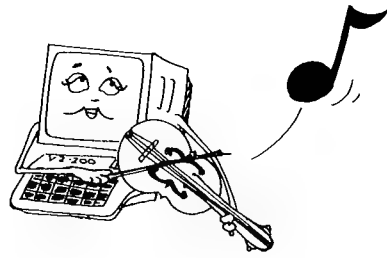
Notes

Chapter 16

A New Way of Storing Variables!

Oh, is that you again? Tremendous...we're really very proud to see that you're still with us. So proud, in fact, that we think a celebration is in order! Before we say another word about computers, there's a little exercise that we'd like you to do.

Ready for anything? OK...follow these instructions carefully:

        1. Stand up
        2. Pat yourself on the back. (Yes, this is rather awkward...but it <u>can</u> be managed if you wriggle about a bit.)
        3. Throw back your shoulders, smile and shout "I've made it!"

    (Oh yes --just one warning. Family, friends and flatmates will find this sort of behaviour a little peculiar...but don't let that stop you! Only another programmer could understand the significance of it all.)

Doesn't that feel good? And don't you think you've deserved it? Please don't be modest--it's great to take credit when it's due. From those first wobbly steps into the world of computers, you've managed to stay on your feet for <u>15 whole chapters</u> of this book!

<u>AND NOW...ON WITH THE SHOW!</u>

Because one thing tends to lead to another, we'll go back to the hotel reception lobby for <u>this</u> chapter, too. And what's next on the "things-to-learn" list? Well...last chapter we talked about a new thing to store in a pigeonhole. Right now, we'll talk about a new way to store them.

    "<u>Another</u> way? But we already <u>know</u> two ways--using <u>LET</u> and INPUT. Surely there aren't any more!"

Don't be too sure...just yet! The hotel clerk in your VZ-200's memory does have one <u>more</u> filing system...one that'll definitely come in handy later on!

You see, both LET and INPUT are very, very useful. But eventually, you might want to store a whole lot of information in your computer's memory. (Especially as your

programs get longer and more complicated!)
Suppose you wanted to create 5 variables--and put something
different inside each one.

You could do it like this....
```
        LET A=5
        LET B=4
        LET C=3
        LET D=2
        LET E=1
```
....but that would take a long time!

So, let's try something completely different!

...Using just two lines:
```
        DATA 5,4,3,2,1
        READ A,B,C,D,E
```

Urk! That's probably left you rather in the dark! Allow us
to shed a little light on the mystery.

Long ago, we mentioned a very important rule about
variables. Remember this?:
> The VZ-200 won't store anything in a memory
> pigeon-hole, until we give it a name to mark that
> compartment with!

So far, we've always provided a name for a variable before
telling your computer what to store inside it. This time
around, however, things are going to be a little different!

The first line in the pair above contains 5 items (in this
case, they're numbers). Typing this line is like giving
your hotel desk-clerk an armful of parcels--and saying:
> "Here--look after these for now. Later on, I'll
> tell you where to store each one."

Hmm...those parcels will have to be go somewhere out of the
way. Like up on a shelf, for example!

> A line beginning with a **DATA** statement is
> exactly like a "parcel shelf"! It's a place for
> things to be kept, while they're waiting to be
> put in their place. (By the way--did you notice
> the commas on the **DATA** line? They act as
> "dividers"--and we use them to separate items
> on the line)
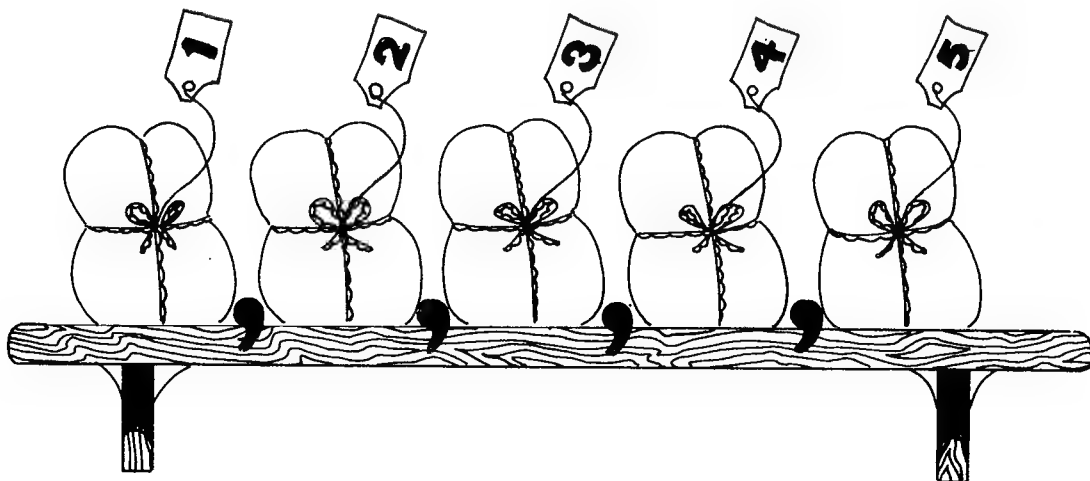
When it's time to take parcels down from the shelf, the VZ-200 always starts at the left, and works along to the right. Computers prefer to do things in this order...they're really very fussy people. (Er, we mean machines!)
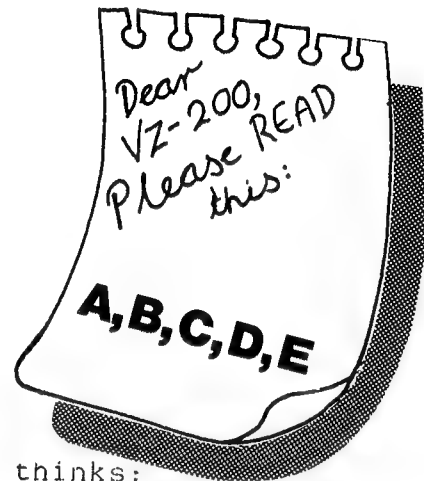
Now—for the second of those mysterious lines. The one beginning with READ.

This line is actually a "list" of variable names. We want the VZ-200 to use each name—one by one—to label an individual memory compartment. Once again, the commas are there to separate each item from its neighbour.

When your VZ-200 sees the READ list, it thinks; "Oh, good. Now I can start sorting out the mail!"

Psst...let's look over the VZ-200's shoulder, and see exactly how this filing system works!

The name at the very top of the READ list is A. So: your computer uses that name to label one pigeon-hole. It then takes the first parcel down from the DATA shelf, and stores it in compartment A!

Right—now for the second name on the list. This is B—and your computer will store the second parcel from the shelf in a pigeon-hole called B.
Ah...now, have you guessed what's going into pigeon-hole C? Exactly—the third parcel! And so it goes, until the last variable name on the READ list has been used.

See what an efficient system this? The VZ-200 will keep working its way along the READ list—one name at a time. And it stores the next parcel from the DATA shelf into each new compartment it labels.

By the way, you don't have to use every single bundle of data on the shelf. Suppose, for example, there are 6 items on your DATA line...and only 5 names on your READ list. The computer won't mind that at all. It will simply store the first 5 packages in its memory—and ignore the "leftovers"!

If we had made those two lines into a little program, we could then add another line asking the VZ-200 to

PRINT A;B;C;D;E

...And when we ran our program, here's what would appear on the screen:

5   4   3   2   1

Get the idea? If you'd asked the your computer
to PRINT E;D;C;B;A, we would've seen

1   2   3   4   5

on the screen, instead!

Unfortunately, there's one thing that your computer will mind. You must make sure that you've got enough bundles of data to store in the compartments being named. Like to see an example? OK--replace the second line of our example with this one:

                20 READ A,B,C,D,E,F

Now, try RUNning it again!

At first, everything will work just fine. Until your computer gets to the last variable name on the list. It'll quite obediently label a compartment as F. But when it tries to take another parcel from the data shelf...oh dear. The cupboard is bare! You've named 6 pigeon-holes--and there were only 5 bundles of DATA. Nothing's left to store in F.

If something like this happens, your VZ-200 will say: "Hey...how do you expect me to do that? Can't you see that I'm OUT OF DATA?"

            The moral of the story:
            Never have more names on your READ line, than
            items on your DATA line!

(If you need to use more than one DATA shelf to store all your parcels, go right ahead! You can have as many as you like.)

Here's another useful piece of information: If you like, you can put just one variable name (say D) on the READ line. Then, you can keep looping the VZ-200 back to the beginning of the program. (How? By using a FOR-TO-NEXT loop, of course!).

Every time your computer comes across the READ line, it will replace the parcel ALREADY in D with the NEXT bundle on the DATA shelf!

And how do you work our how many times the VZ-200 needs to loop around? Easily! Your computer simply needs to make one loop for every item on the DATA line.

To prove to you just how useful this new system can be, we'd like you to type in the following sample program. But first, let us introduce you to an imaginary friend of ours -- Mr B. Careful.

Now, Mr Careful is a shopkeeper. And like most business people, he sometimes needs to check whether a customer is reliable. (That is, whether they can be trusted to pay their bills!).

This program is designed to do the checking for him! You see, Mr Careful gives a special "serial number" to every one of his customers. And "listed" in this program are the serial numbers of customers with a <u>bad record</u>.

Mr Careful can INPUT any serial number that he wishes to check...and the computer will tell him whether or not it is on the "bad list"!

Sounds rather clever, doesn't it? Let's see if it works!

```
5 CLS
10 DATA 11067, 12549, 22871, 57824, 86256
20 INPUT "CUSTOMER NUMBER";N
30 FOR L=1 TO 5
40 READ D
50 IF D=N THEN GOTO 90
60 NEXT L
70 PRINT "THAT NUMBER IS FINE BY ME!"
80 END
90 PRINT "NO! THAT NUMBER IS BAD!"
100 END
```
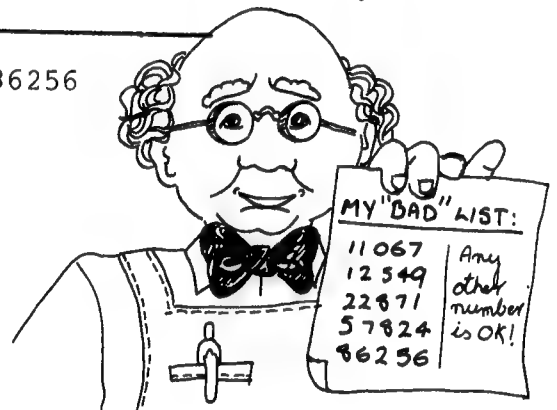
Confused? Never mind -- the whole thing is really very simple to analyse.

Line 5:
Easy enough! It's just giving us a nice, clear screen to RUN on.

Line 10:
Our DATA shelf! There are 5 items stored here. And remember, each "package of data" is actually <u>the serial number of a customer who hasn't paid their bill!</u>

Line 20:
Hey...we haven't encountered a line like <u>this</u> before! (No, we haven't. Sorry 'bout that...just trying to pull a fast one on you!) This is really just an ordinary INPUT statement. But we've given it a little extra "flourish"--by slipping in a line that reminds the user just what they're supposed to be INPUTting.

Line 30:
Here's the start of our "reading the data" loop. Because there are 5 different items on the DATA line, we'll need to loop around the same number of times! (We haven't given it the name L for any special reason, except that L stands for loop!)

Line 40:
    We've only just finished telling you what this one
    does! The first time around, the READ line asks the
    VZ-200 to store the first parcel from the DATA
    shelf in a pigeon-hole marked D. The second time
    around it stores the second parcel...and so on.

Line 50:
    Using dear, faithful IF-THEN, this line puts the
    customer number in N to the test. IF it is equal to
    the number stored in D, THEN your computer will
    GOTO line 90. Line 90 prints a message to inform
    you that yes, it is on the "bad number" list! And
    straight after this line, comes line 100 -- which
    ENDs the program.
    But if N is not equal to D, the VZ-200 will simply
    drop down to the next line of the program...

Line 60:
    Here's the NEXT statement to close the loop. It'll
    send the computer back to the beginning of the
    FOR-TO-NEXT loop. Then it will replace the number
    already in D with the second parcel from the DATA
    shelf, ready to be compared with N. This line will
    keep looping the VZ-200 around--until it's made its
    5th trip. And when it has, it's free to go to line
    70.

Line 70:
    If the your computer has made it this far, then the
    number you are testing has passed with flying
    colours! This line will print a statement, to tell
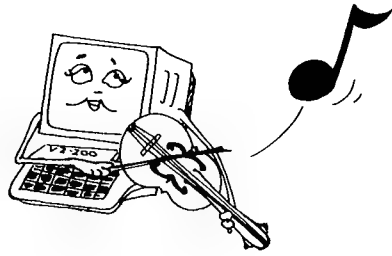   you that the number is OK.

Line 80:
    This one speaks for itself!

Go ahead -- try it for yourself! The first time you RUN the
program, invent an imaginary "customer number" to INPUT.

Does it work? Great! Now, try it again...but this time,
INPUT a number that you know is on Mr B. Careful's "bad
list". (Just to be sure that this part works, too!)

Notes

Chapter 17

A Bit More about DATA & READ...

How'd you like a little rest? Thought so! After all those
long chapters, we really think that it's time for something
short, sweet and useful.

So right now, we're going to tell you just <u>one</u> more
(helpful) thing about DATA and READ.

After we'd finished RUNning Mr B. Careful's "checking"
program, what do you think happened to those five parcels?
We know that they were all taken <u>down</u> from the DATA shelf.
And we know that they were stored, one by one, in
pigeonhole D. But what on earth happened to each parcel
when it was removed from D and replaced by the next?

Good question. We haven't told the VZ-200 to put the
parcels back on the DATA shelf...so just at the moment,
they're nowhere in particular!

If we wanted to "check" another customer, we'll have to
restore the parcels to their original positions on the
shelf. To do this we <u>could</u> run the whole program all over
again. But that would be rather tiresome.

Just a moment -- here's a better idea! We also <u>could</u> learn
a new BASIC word...one that'll restore the parcels, without
having to start all over again!

Brilliant! The new word is called (logically enough)
RESTORE. And to use it in Mr Careful's program, we need to
add these four lines:

```
72 RESTORE
74 GOTO 5
92 RESTORE
94 GOTO 5
```

Now, our program looks like this:

```
5 CLS
10 DATA 11067, 12549, 22871, 57824, 86256
20 INPUT "CUSTOMER NUMBER";N
30 FOR L=1 TO 5
40 READ D
50 IF D=N THEN GOTO 90
60 NEXT L
70 PRINT "THAT NUMBER IS FINE BY ME!"
72 RESTORE
74 GOTO 5
80 END
90 PRINT "NO! THAT NUMBER IS BAD!"
92 RESTORE
94 GOTO 5
100 END
```

Guess what it does now! When the computer finds a RESTORE statement, it will put all the parcels back on the DATA shelf -- in their original order!. The GOTO command immediately after the RESTORE line simply says "go to the beginning of the program again".

And once the computer is back at the start once more, we can INPUT another number to be tested.

Why do we need two pairs of RESTORE and GOTO lines? Because if you look carefully at the program, you'll that it has two possible endings! That's why we had to slip a RESTORE and a GOTO command in

●After line 70 (the end of the program if the number being checked is OK), and
●After line 90 (the end of the program if the number is bad!)

Go ahead...try your updated program now!

A little aside:
Did you notice the END statement on lines 80 and 100 of our new program? And did you also notice that they're not really necessary...because the program always loops back to the beginning before it gets to them? You did? Oh, very good. To tell the truth, we only included them because finishing every program with END is a good habit. (Even if sometimes, the statement isn't really needed!)

Notes
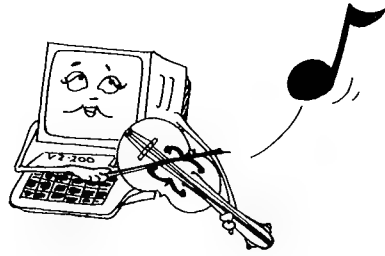
Chapter 18

Music To Your Ears!

Funnily enough, the world's very first musician was someone
that you've already met. (We introduced him at the very
beginning of this book). Yep -- it's our old friend, the
caveman. And the very first musical instrument? That was
the caveman's first invention...the hammer!

Of course, the caveman developed his hammer with only one
purpose in mind: To hit something harder than he could with
his bare hands. (Exactly what he planned to hit remains a
mystery. It could have been rocks...or wood...or enemy
tribesmen!)

But while he was thumping away with his new creation, he
made yet another discovery.

> Every time the hammer hit something, it created...
> A SOUND!

Much to his delight, the caveman found that he liked this
"sound". In fact, he liked it so much that he just kept on
making it. Over and over again.
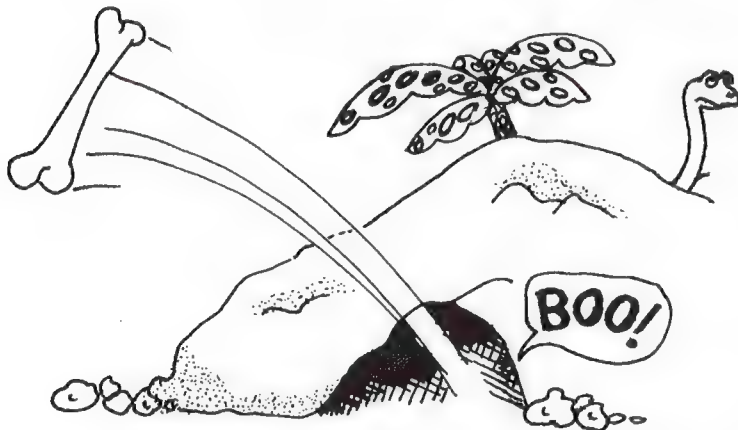
You'll probably agree
that this "tune" wasn't
really destined for the
top of the charts! To
tell the truth, man's
musical future was
looking pretty
bleak...until the people
in the neighbouring
caves complained
(loudly!) that the noise
was BORING!

BOO!

Luckily, it's hard to keep a good caveman down! Our
prehistoric friend wasn't discouraged -- he simply began
experimenting once more. And before too long, he'd stumbled
upon another discovery. He found that by whacking different
objects with his hammer, he could vary the sounds he
created!

    For example, he discovered that:
        Hitting a boulder made...a low sound
        Hitting a log made...a medium sound
        Hitting a pebble made...a high sound
        And hitting a melon made...an awful mess! (All
        inventors make mistakes)

The frequency of each sound (that is, how "high" or how "low" it was) came to be known as its pitch.

By arranging these sounds (today we call them "notes") in different patterns, man was able to create a very pleasing effect. In fact, he was so pleased that he felt these patterns deserved a special name. He decided to call them "MUSIC"!

Years...and centuries...and ages rolled by. As time passed, man discovered even more ways to make his music more interesting. And here's one of the most important! He learned to vary not only the pitch of the notes, but the length of time that each sound lasted for! This is called the "duration" of the note.

And thanks to pitch and duration, man was able to arrange notes in much, much more intricate patterns. Imagine how proud he was of them! There was only one problem -- some of these musical designs were so intricate, they were almost impossible to remember!

Man just hated to waste his masterpieces -- but unfortunately, he didn't have a good enough memory to remember his tunes for very long. Eventually, it occurred to him that to preserve his musical patterns, he'd have to invent a way of "writing them down".

What a problem! It certainly must have puzzled him at first! (After all, how can you "write" a sound?)

But being a patient species, he persevered...until at last, he came up with a very bright idea indeed! He decided to use a special "picture language" to represent different sounds.

Today, we call this language musical notation! (Think about it...at some stage, just about everyone has seen a piece of music "written down" on paper.) If you don't know much about music, it probably looks like a whole lot of tadpoles sitting on five telephone wires!

But if you do know something about musical notation, you know better than that! You'll be able to tell us that those little shapes aren't tadpoles at all...they're actually "drawings" of individual musical notes.

And it's not a set of telephone wires that the notes are sitting on, either! In musical terms, these five lines form something called a "stave".

So, exactly how does this system work? Quite simply, really.

The pitch of a note is determined by its position on the stave,
...and...
The duration of a note is determined by the shape used to represent it!

94

Slowly but surely, man organised his notes into something
he called a musical "scale". This was a series of 8 notes.
Why did he call it that? Simply because each note "climbed"
one step higher than the last one...in exactly the same way
that a person might climb -- or "scale" -- a ladder!

When a musical scale is "sung", it goes like
this:

DO-RE-MI-FA-SØ-LA-TI-DO

Each note in the scale had its own position on the stave.
Man gave a name to each of these positions, using a letter
of the alphabet to label each one.
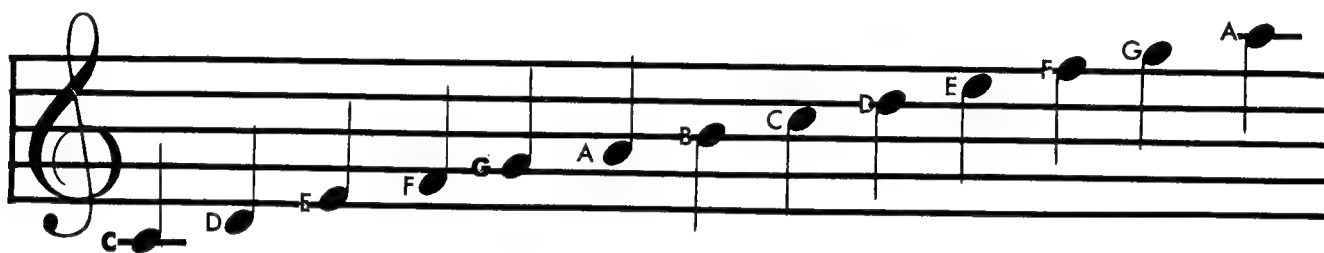
This "musical scale" forms a sort of basic building block!
We can add scales together by overlapping the last note of
each scale. By doing so, we can produce a series of sounds
that climb higher and higher -- or lower or lower.

DO-RE-MI-FA-SO-LA-TI-DO-RE-MI-FA-SO-LA-TI-DO

This "adding" could go on and on, really! So, to avoid
getting lost among so many different notes, man decided to
use one note as a landmark. This way, he'd always know
where he was!

The note that he chose as his landmark is
called Middle C. Any notes that are lower
than Middle C are said to be in the "bass
clef".
At the beginning of a staff containing
these low notes, you will see a symbol
that looks like this:

All the other notes -- the ones higher
than Middle C -- are said to be in the
"treble clef". If the notes in a staff
are higher than Middle C, the staff will
begin with a symbol like this one:

(What's that you say? This symbol looks a lot more
familiar? Well, that's because it's a whole lot more
common. Most pieces of music that you see will begin with
this "treble" sign!)

By the time man reached this stage in his musical
development, his caveman days were millions of years behind
him. And to tell the truth, he was beginning to feel pretty
smart! So smart, in fact, that he began hunting for ways to
improve his musical scale.

95

To give himself an even bigger range of sounds to choose from, he slipped more notes between some of the notes already on the scale. He called these notes "sharps"...and he placed them one half-step up from existing notes. A note that is to be played "sharp" will have the same name as the main note one half-step below. And it will have the little # symbol next to it.

> As you can see, our system of writing music is pretty efficient. It allows us to think of almost any sound we like, and write it down on paper!

Now, this system works just fine for humans! When a musician wishes to play a tune (that is, a series of sounds) that's written down on paper, he merely has to look at:

> 1. The position of the notes, and
> 2. Their shape

...and he will know exactly which sounds to produce!

Your computer, unfortunately, has a bit of a handicap when it comes to "reading" music. For obvious reasons! You see, to look at something we humans use two special pieces of equipment -- our eyes. And the VZ-200, of course, doesn't have any! Anyway, even if it could look at the music, the VZ-200 wouldn't be able to understand our picture writing system. (Computers don't think that way.)

So, because the VZ-200 can't see the song, we'll have to find a way to describe each note to it!

The easiest way to do this is to use a system of number codes.

> As we've mentioned, there's a pair of things that a musician needs to know about a note, before he or she can play it. To describe a certain sound to your computer, these two things must be "represented" by a pair of numbers!
>
> The first thing you'll need to tell the VZ-200 is: The position of the note on the stave (it's pitch)
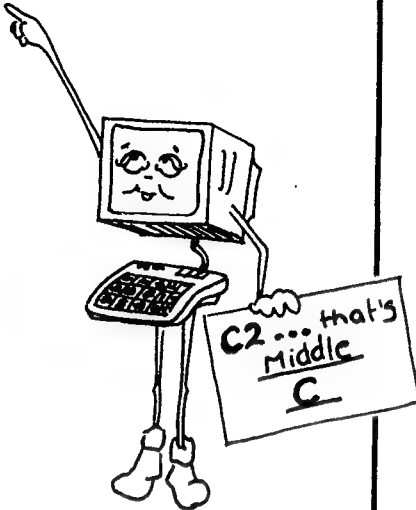
The VZ-200 can play 31 different notes (or pitches)...plus a "rest". (That's a short break in the music). In musical notation, it's represented by a variety of symbols (different shapes, for rests of different lengths!)

Each of these notes has a code number all it's own. The entire list of notes, and their code numbers, is shown on the next page:

Whole _____ 1/2 _____ 1/4 _____ 1/8

Code:	Name of note:	Code:	Name of note:
0	rest	16	C2
1	A1	17	C#2
2	A#1	18	D2
3	B1	19	D#2
4	C1	20	E2
5	C#1	21	F2
6	D1	22	F#2
7	D#1	23	G2
8	E1	24	G#2
9	F1	25	A3
10	F#1	26	A#3
11	G1	27	B3
12	G#1	28	C3
13	A2	29	C#3
14	A#2	30	D3
15	B2	31	D#3

The notes on this side of the table describe notes below Middle C. So we'd use them for a piece of music beginning with a bass clef

The notes on this side of the table describe notes above Middle C -- and we'll use them for music beginning with a treble clef

To describe the pitch of a note to your computer, just look up the code number that corresponds to it. This number will be the first half of your pair.

Now, you've "described" the note's position on the stave. What else does your computer need to know about the sound?
Right! It must also be told

The shape of the note (its duration)

...so your computer will know how long the note should last!
As you might have guessed, there's a special code number for each different shape of note:

DURATION Code:	A note shaped like this:	Lasts for this length of time:
1	♪	⅛ of a note
2	♪	¼ of a note
3	♪.	⅜ of a note
4	♩	½ of a note
5	♩.	¾ of a note
6	♩	1 full note
7	♩.	1½ notes
8	♩	2 notes
9	♩.	3 notes

This number will be the <u>second</u> half of your description.

Well, there you have your pair of instructions. Together,
they form a description of a single note.

> Not so hard, is it? Simply by using the two tables
> above, you should be able to "describe" pieces of
> written music to your computer! Now, we certainly
> don't suggest that you try to teach it to play
> Beethoven's 5th Symphony! (the VZ-200 isn't <u>that</u>
> cultured!) Please, stick to simple little tunes.
> They're really just as much fun! A very good place
> to find them is in a children's song book.

To "translate" a piece of written music into a language
that your computer <u>can</u> understand, you have to:

> 1. Decide where each note is on the stave, and look
> up the code number of that position on the "pitch"
> table.
> 2. Decide what shape each note is, and look up the
> code number for that shape on the "duration" table.

The piece of music you'll describe to your computer --
instead of a series of different shaped notes with

different positions on the stave -- will be a series of
different code-number pairs!

Oh, yes! We almost forgot something very important! All
those number pairs won't mean a thing to the VZ-200...until
you tell it just what they are for! So, in front of every
pair, we'll need to put some sort of signpost. One that
tells your computer "These two numbers represent a musical
note. Please play it for me!"

> To do this, we must learn yet another BASIC
> word. And what's it called? You guessed
> it...SOUND!

The SOUND command goes in front of every pair of code
numbers.

Here's an example:

*The code for this note's pitch is 28,
and the code for its shape is 2 --
so to describe it, we'd say*
SOUND 28,2

*The code for this note's pitch is 23,
and the code for its shape is 6 --
so to describe it, we'd say*
SOUND 23,6

*The position of the note's "tadpole tail" isn't
important...it can be hanging down below the note, or
standing straight up above it. What does matter is whether
note is "filled in" or "hollow"; whether there are any
little "flags" hanging on its tail; or whether it has a dot
right next to it.*

From Sounds..to Songs!

Teaching your VZ-200 to sing.

Now that you know how to "describe"
individual notes to the VZ-200,
there'll be no stopping you! To
celebrate your new-found ability, let's
try something really exciting. Let's
try to teach your computer to "sing us
a song"!
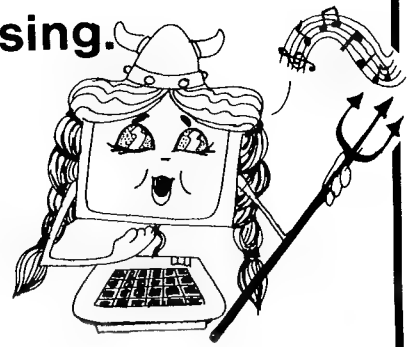
Hmmm...we'd better choose something simple. How about a
good old nursery rhyme? "Twinkle, Twinkle Little Star"
would be ideal.

Now, what must we do before your computer can perform for
us? That's right...we'll have to give it a description of
every single note in the song! To do it, we'll use:

Twin-kle, twin-kle, lit -tle star, How I won-der

what you are! Up a-bove the world so high,

Like a dia-mond in the sky. Twin-kle, twin-kle,

lit-tle star, How I won-der what you are!
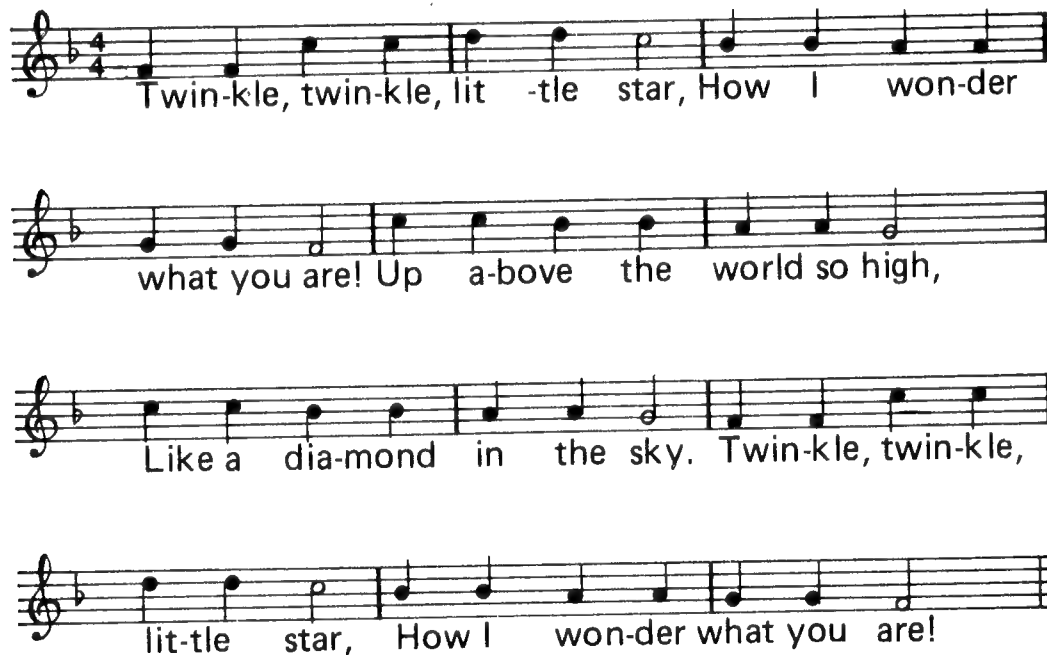
If you count each little note, you'll find that there are
42 of them in this tune. Good heavens! Does that mean that
we have to use 42 SOUND commands in our program?!

No, calm down...there's an easy way out of this. All those
SOUND commands aren't necessary at all. Instead, we can use
a really clever trick -- one that we've already taught you!

Have you guessed which clever trick we mean? (After
all, we've learned so many)! Here's a clue -- there
are two BASIC statements involved (and they sort of
go hand-in-hand!). When we first mentioned them, we
said that they're very useful for storing a lot of
information in your VZ-200's memory.

If you guessed DATA and READ, you got it right first try!
These two statements will certainly save us a lot of
unnecessary work. (Because we sure have got a lot of
information to store!). Like to see how? OK -- very
carefully, enter the program below.

```
5 CLS
10 DATA 21,4,21,4,28,4,28,4,30,4,30,4,28,6,26,4,26,4,25,4
20 DATA 25,4,23,4,23,4,21,6,28,4,28,4,26,4,26,4,25,4,25,4,23,6
30 DATA 28,4,28,4,26,4,26,4,25,4,25,4,23,6,21,4,21,4,28,4,28,4
40 DATA 30,4,30,4,28,6,26,4,26,4,25,4,25,4,23,4,23,4,21,6
50 FOR L=1 TO 42
60 READ P,D:SOUND P,D:NEXT L:END
```

Done that? Good. Now -- before you have a chance to feel
confused -- we'll explain exactly what this terrifying
jumble of numbers is (and how it works!)!

Analysing the program

> Lines 10, 20, 30 and 40:
> As you can see from the DATA statements at the
> beginning, we're using these four lines as "storage
> shelves". (We need four of them...because we've got so
> much DATA to store!)
> And exactly what are we storing? Pairs of code numbers!
> Each pair is a description of one note of our song.
> (The first number in the pair describes the pitch of
> the note. The second number describes the duration.)
> Just count them for yourself...you'll find that
> altogether, there are 42 pairs on our DATA shelves.
> > For example:

> > 10 DATA 21,4 , 21,4 , 28,4 , 28,4 , 30,4
> >   ...and so on.

So! We want the VZ-200 to READ these numbers -- two by
two -- and put each one into a variable pigeon-hole.
Once our two code numbers are safely stored, we can use
just one SOUND command, followed by the pair of
variable names!

The computer will play the note described by whatever
two numbers happen to be in the pigeon-holes!

> Line 50
> Our "counting" loop! Because there are 42 pairs of code
> numbers, we must ask the VZ-200 to loop around 42
> times. (Each trip around the loop will replace the
> numbers already in the pigeon-holes, with the next pair
> from the DATA shelves!)

Line 60
Aha..this is one of those clever lines containing more
than one command. Remember, it's OK to do this -- as
long as each one is separated by a colon!

First command: Here's our READ statement. It will store
the code numbers, pair by pair, in two pigeon-holes.
The first one is named P (this is where the pitch code
goes); and the second one is named D (the duration code
is stored here!).

Second command: Asks the computer to play the note
described by P and D.

Third command: Sends the VZ-200 looping back to store
the next two code numbers in P and D.

Fourth (and last!) command: ENDS the song when the
VZ-200 has made its 42nd loop.

Have you finished typing the program into your VZ-200?
Well, it would be a good idea to LIST it -- just to be
sure that you've got every single number right. ('Cause if
you haven't, you might hear your computer sing a bit "off
key"!)

        Everything perfect? Great! And now, the big moment
        has arrived...it's time to RUN the program.

Ahh...isn't that wonderful? If you think it deserves a
standing ovation, go right ahead! And if you'd like an
encore, simply RUN the whole thing all over again!

            Now, for an interesting (although totally
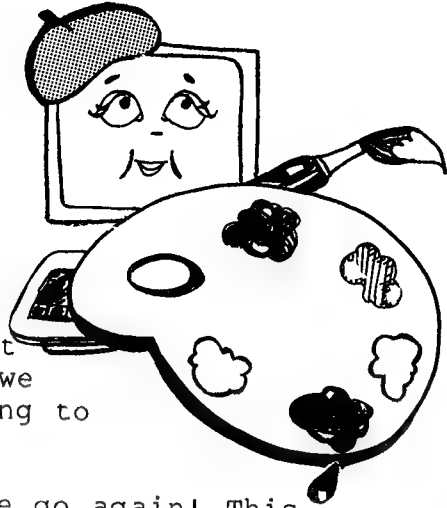   useless) piece of information:
            Your VZ-200 has just achieved more than you
            could ever imagine. Not only has it played you
            a song...but it has played you a song written
            by a very, very famous composer!

            Yep...believe it or not, "Twinkle, Twinkle
            Little Star" was composed by none other than
            Mozart!

Chapter 19

Great fun with Graphics!

Will this little computer of yours ever
cease to amaze us with its abilities? It
seems that for every incredible talent we
learn about, there's another just waiting to
be discovered.

So, hang onto your hats...'cause here we go again! This
chapter, (believe it or not) we'll unearth your VZ-200's
hidden artistic ability!

Now, don't get us wrong. If computers had been around in
the days of Leonardo da Vinci, the great man wouldn't
really have had much to worry about! We are going to learn
how to "draw" on your VZ-200's screen. But don't expect to
produce another Mona Lisa!

The art of creating pictures or patterns with your computer
goes by the odd name of "computer graphics". Mastering the
art takes time, and patience. But boy -- it sure is fun!

Your computer has 2 different "Modes" that it can be in to
produce graphics. The first is plain old Mode (Ø) -- and
your VZ-200 enters it just as soon as you turn it on!

Mode (Ø) is the one we use for putting keyboard characters
on the screen.

And what, exactly, are "keybord characters"? Quite simply,
they're any character (or command) that you can find on
the VZ-200's keyboard! Including those funny little shapes
called "graphics characters"!

These different "graphics characters" are rather like
tiles...the sort you might see on a bathroom wall! Each one
has a different design (there are 15 of them, altogether).
By arranging the tiles in patterns on the screen, we can
create...a mosaic! (Well, sort of!)

Unfortunately, the tiles (or "picture building blocks", if
you like) are rather, well, large and lumpy. Which means
that whatever pictures we create with them will be large
and lumpy, too!

As we said, "computer graphics" seems a rather
strange name. Or does it? If you look very closely
at the word graphics, you just might find another
word lurking in there. One which describes the
whole idea very well indeed!

Give up? OK -- the word is graph. And it's a very useful
description, because your VZ-200's screen is actually
ided up into an imaginary graph!

<u>In Mode (0), the graph is 64 blocks wide and 32
blocks deep</u>

One keyboard character takes up 4 of these blocks...and
that's why we can only produce rather "coarse" designs in
text mode.
Wouldn't it be lovely if we could divide the screen up into
much smaller blocks? That way, we could create patterns
that weren't quite so clumsy!

Yes, it <u>would</u> be lovely. And yes, we <u>can</u> do it! It's just a
matter of changing the VZ-200 over to our second
mode...<u>Mode (1)</u>!

        Easier done than said! Just type MODE (1) and
        press RETURN .

        <u>In MODE (1), the imaginary graph is 128 blocks wide
        and 64 blocks deep...which is a whole lot finer!</u>

When you're in MODE (1), things work rather differently!
For starters, we <u>can't</u> type any character from the keyboard
onto the screen!

        "That's silly!" you complain. "If I <u>can't</u> put
        anything onto the screen, what use <u>is this</u> Mode
        in the first place?!"

Now, now...calm down. We didn't say you can't use the
screen -- <u>we just said you can't use the text characters!</u>

Instead, when we want to put something onto the screen,
we'll use another BASIC command. This time, it's one called
SET.

        SET asks your computer to "colour in" just <u>one</u>
        of those little blocks in the graph.

Each block has it's very own position on the screen. So,
before we can tell the VZ-200 exactly <u>which</u> dot to SET,
we'll have to give it some <u>"co-ordinates"</u>.

        "<u>Co-ordinates</u>? Help! This is getting a bit too
        <u>technical!</u>"

Please, don't let that term scare you off! 'Cause you're
probably more familiar with co-ordinates than you realise.
Just stop for a moment, and think about the way you look
something up in a street directory. There's certainly
nothing tricky about <u>that</u>, is there? All you have to do is
look up the name of whatever you're trying to locate in the
Index. Next to the name, you'll see a pair of figures
(usually, one is a letter and one is a number).

Like your VZ-200's screen, most road maps are divided up
into a graph...with one set of lines running horizontally,
and one running vertically. One of that pair of figures
will be the name of a horizontal line; and one will be the
name of a vertical line. To find your target, all you have

to do is trace along those two lines. And at the spot where those two lines collide? Sure enough, there's the street you were looking for!

Now, listen closely...those two figures were actually a pair of co-ordinates!

And SETting a point on your computer's screen is really no more difficult! You simply need to stype SET...and then give your computer a pair of co-ordinates! The first one will be the name of a vertical line in the graph (and it can be called anything from 0 to 127). And of course, the next one will be the name of a horizontal graph line! (It's name can be any number from 0 to 63.)

What if you want to get rid of a block that you've coloured in using SET? No problem! We can use another BASIC statement -- one that's the exact opposite of SET!

It's called RESET, and you need to follow this command with the same co-ordinates that you used to SET the block with!

But gosh...SETting just one little block at a time is a bit tedious! It's going to take us hours (and lots of SET commands) to colour in a decent area of the screen, isn't it?

Well, actually, it isn't! We can use a very neat little trick to save ourselves all that time and effort.

> To tell the truth, this trick isn't even a new one. We've used something like it before...when we taught the VZ-200 to "sing" us a song!

Once again, we'll use just one SET command...followed by two variables. In the first variable pigeon-hole (let's call it V), we'll store a VERTICAL CO-ORDINATE. And in the second pigeon-hole? That's right -- we'll use it to store a HORIZONTAL CO-ORDINATE! (Let's call this one H.)

Want to see how it works? OK...type in this program.

```
5 CLS
10 MODE (1)
20 FOR V=0 TO 127
30 FOR H=0 TO 63
40 SET(V,H)
50 NEXT H
60 NEXT V
```
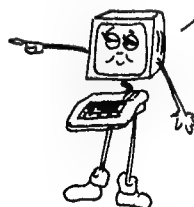
SEE THE "NESTED" LOOP?

Analysing the program

Line 10 puts us in the right Mode (that's Mode (1)!)

Line 20 is the beginning of a FOR-TO loop. Every time around, the value of V will increase by 1.

On line 30, we open another FOR-TO loop! Like the first loop, it will increase the value of H by 1 on each "trip"!

Line 40 -- and here's our SET command. It will SET the point described by whichever co-ordinates are stored in the pigeon-holes.

Line 50 says NEXT H...and of course, it closes one of our FOR-TO loops. Remember, the H loop is "nested" inside the V loop...so we MUST close this one first! When the computer finds this line, it will zoom back to line 30 -- the start of this particular FOR-TO loop -- and increase H by 1.

The VZ-200 will keep looping around until the value of 63 is stored in H. Then it will drop down to:

Line 60 -- which says NEXT V. This marks the end of our "outer" loop. So the computer will zoom right back to the beginning of this loop (that's line 20)...and increase the value of V by 1!

That's it! Are you ready to try it out? Good...just type RUN.

Goodness! Look at that -- the VZ-200 has started to "paint" it's entire screen! (Be patient, though...it'll take quite a long time to finish it!)

Want to get rid of the "paint-job"? It's easy enough to do! When your computer has finished, just change line 40 to say
                    RESET V,H
When you run this updated program, you should see the VZ-200 wipe off the paint off the screen!

Making Rainbows!

(It's not called a "colour computer" for nothing!)

Something about your VZ-200 has probably been puzzling you, ever since you met it. And that "something" is the row of coloured "markers" on the keyboard! (Do you see them? They're above the first eight keys in the very top row.)

The marker above the key that we use to type 1, is green. Above the 2 key is yellow, and above the 3 key is blue. Next comes red, then buff, then cyan (that's another name for light blue!), then magenta...right up to the 8 key, which has an orange marker above it.

Yes, they do look very pretty. But they're certainly not there for decoration alone!

The number on each key just happens to be the code for that particular colour. We use these code numbers, together with a brand new BASIC statement (now, be patient -- we'll learn what it is in a moment), to produce colours on your VZ-200's screen!

By now, you've almost certainly noticed how logical the names of most BASIC commands are! In fact, we hardly need to tell you what our new statement is called. Surprise, surprise...its name is COLOR!

Rather like the SOUND statement, COLOR is always followed by a pair of code numbers.
- The first number describes the foreground colour of the screen, and
- The second number describes the background colour.

Remember the two different "Modes" we told you about? Well, when it comes to colour, each Mode has a slightly different set of rules!

MODE (0)

In this mode, you have a choice of two different background colours. The first one is GREEN. And when it's being used as a background colour, it has a special code number -- 0. (We're all very used to seeing a green screen...'cause the VZ-200 automatically chooses this colour whenever you turn it on!)

For example, to tell your computer that you wanted the screen to have a green background, and a blue foreground, you would use a command like this:
COLOR 3 , 0

In this case, 3 is the code for blue...so it comes first.
The (background) code for green is Ø...so it's the second
number in the pair. (By the way -- did you notice that we
use a comma to separate each code number?)

There is an alternative, though. The second choice for
background colour is ORANGE...and its code number is 1.
Hmmm...a bit of variety might be rather nice! Let's try it
out -- using our new COLOR command.
Oddly enough:
Because we're only worrying about background colour for the
moment, we can leave out the first (foreground) code
number...and just say

<div align="center">COLOR,1 <u>RETURN</u></div>

(Of course, the space where the foreground colour code goes
is empty. But we STILL need to type the comma that would
usually follow it! This tells your computer that you're
giving it a code for the background colour only.)

> Are you sure that you're ready for this? Even
> though it's so simple, this is probably the
> most spectacular thing we've tried yet! Now, if
> you think you can stand the excitement, try
> typing COLOR,1.

Kapow! Have you ever seen anything so orange? We warned you
that it was quite a change! And switching back to familiar
old green is just as easy, too. Simply type COLOR,Ø...and
there it is!

> Ah, isn't power a marvellous thing? Let's get
> smarter still...and try bringing some foreground
> colour into the picture as well! How about a red
> foreground, and a green background?

Agreed? Right -- just type:

    COLOR 4,Ø

> "Hey! I see the green...but where's the red? It
> didn't work!"

Oh, come now...would we give you a "dud" command? Of course
not! As a matter of fact, it did work. There's a very good
reason why you can't see any foreground colour. And the
reason is that there's nothing on your screen but text.

> The ONLY keyboard characters that CAN change
> colour, are the GRAPHICS CHARACTERS!

Just try typing in some graphics characters...you'll find
that they're red as can be!

MODE 1

In this mode, your colour choice is a bit more limited.
(Sorry 'bout that -- you can't have everything!)

Your background colour can be either GREEN (and its code
number is still Ø) or BUFF (code number 1).

GREEN	YELLOW	BLUE	RED	BUFF	CYAN	MAGENTA	ORANGE
1	2	3	4	5	6	7	8
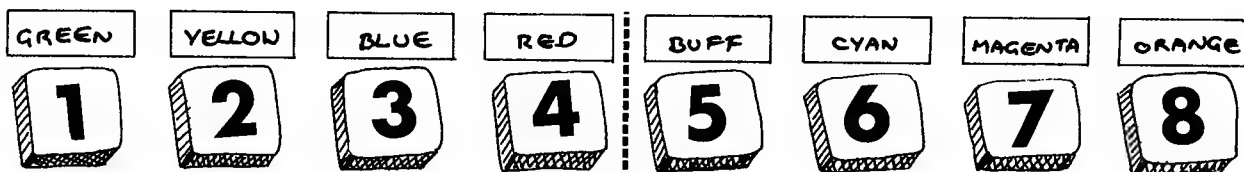
If you choose a green
background, you can
only use this half of the
VZ-200's colour range...

...and if you choose
a buff background, you
must stick to this half
of the colour range!

Now, we'd hate to state the obvious! But just in case,
there's one thing we'd better point out:

To get the COLOR statement to work, you must have a
colour TV!

Chapter 20

Our Very Last Words!

And now, as the sun sinks slowly in the west (or whatever usually goes with a happy ending!), we draw our little book to a close.

Have you enjoyed your first little foray into the world of computers? We certainly hope so...because we sure have enjoyed showing you the way.

And what a long, long way it's been! Ever since that fateful moment when you first opened this book, we've been meeting challenges and taking them in our stride.

For old time's sake, let's look back at some of the hurdles we've managed to clear:

- "What? You expect me to use this computer? I don't even know what the darn thing is!"
    For some of you, this was probably one of the biggest worries!

- The computer/human language barrier.
    Do you realise that you are now "bi-lingual"? You can now communicate -- almost fluently -- in BASIC! (Truly! Just think of all the words you've learned.)

- That big, bad thing called "computer programming".
    Remember the first time we mentioned this one? It probably scared most of you silly. And yet here you are, ready to start writing your own programs!

Whew! They were biggies..and that's just a few of them! (In fact, after so much practice at clearing hurdles you're probably ready for the Olympics!)

The world of computers is very wide indeed. Far too wide to fit into a nutshell (or into one little book!). We could never have hoped to cram it all into just 114 pages!

What we could hope to do was provide you with a basic knowledge of computers in general--and the VZ-200 in particular.We want you to be well-equipped to continue your developement as a programmer. And if this manual has achieved that, at least, we couldn't be happier!

Remember that a good, strong friendship costs nothing...except, of course, TIME. So to become really well acquainted with your VZ-200, be prepared to meet the expense! Take time to experiment...time to practise...and time to play games. (We promise you--it'll be worth the effort!

And now...a last piece of fatherly (er, or motherly?)
advice. If ever you're in trouble, never be afraid to GO
BACK TO THE BASICS! Is something going wrong? Anything at
all? Just come back to this little manual.

We'll always be right here--ready to explain it all over
again!

Also: don't be afraid to experiment! Trial-and-error is one
of the very best ways to learn...and you can't possibly
damage your VZ-200 by trying your ideas on it!

So! Goodbye for now...and GOOD LUCK!

Appendix A

How to make your VZ-200 love you!

It's not hard...really it's not. Just read the little hints below. They're very good advice...and if you take it, your computer will thank you for it!

●Handle with care!
Being dropped is a rather traumatic experience for the VZ-200. So please, don't bash it, bump it, or use it to prop the door open! You wouldn't work too well if somebody did that to you--and neither will your computer.

Heat bothers your computer, too. Be careful not to leave your VZ-200 on the window-sill, the patio, or anywhere the sun is strong. No, it won't get sunburnt...but it will get sick.

●There's no place like home...
...and to your computer, "home" is the place (a safe one, naturally) where you put it when you've finished using it! If you leave it connected to the power point all the time, the VZ-200 will get rather hot and bothered.

●A little etiquette, please!
Unlike a human, the VZ-200 won't mind if you eat in front of it. But please -- remember your table manners! 'Cause your computer will be offended if you slop orange juice, and sprinkle crumbs, all over its keyboard! Really, it's a good idea to keep food and drink away from the VZ-200. (Just to be on the safe side.)

●Choose a safe spot.
If you were looking for a place to set up your VZ-200, you certainly wouldn't choose the middle of a busy highway! Unfortunately, some areas in most homes can be almost as hazardous! So, when you settle down for a long session with your computer, try to pick a spot that's out of everyone's way. (Believe us, there's nothing more frustrating than having someone trip over the power cord!)

●Don't play doctor!
If you were sick, would you let one of your friends operate on you? No -- we didn't think so. When humans get sick, we go to a people-doctor...and when computers get sick, they need to go to a computer-doctor! So, if your VZ-200 isn't working properly (and you're sure that it's not because of something you're doing wrong), just bring it along to a Dick Smith store or dealer. One of our very capable computer-doctors will soon have it back in action. Please -- don't try to perform any "operations" all by youself!

Appendix B

The BASIC words we've learned

Chapter 4

PRINT (used with quotes)
GOTO   END   LIST   RUN
BREAK   CONT   CLS   NEW

Chapter 5

PRINT (used without quotes)

CHAPTER 6

LET

CHAPTER 7

INPUT

CHAPTER 8

IF-THEN   ELSE

CHAPTER 1Ø

RND   SQR

CHAPTER 12

FOR-TO   NEXT

CHAPTER 13

STEP

CHAPTER 14

GOSUB   RETURN

CHAPTER 16

DATA   READ

CHAPTER 17

RESTORE

CHAPTER 18

SOUND

Appendix C

Reserved words

When it comes to labelling variable pigeon-holes, these words are a NO-NO! The VZ-200 has "reserved" them for special purposes...and if you try to use them for something else, you'll cause confusion.

> REMEMBER: You mustn't use:
>
> ● Any word on the reserved list, or
>
> ● Any word whose first two characters are the same as the first two characters of a reserved word!

(Why? Because the VZ-200 doesn't look past those first two characters of a variable name. To your computer, DATA and DAVID are the same name!)

ABS   AND   ASC   ATN

CHR$ CLOAD   CLS   COLOR   CONT   COPY   COS   CRUN   CSAVE

DATA   DIM

ELSE   END   EXP

FOR

GOSUB   GOTO

IF   INKEY$ INP   INPUT   INT

LEFT$ LEN   LET   LIST   LOG   LLIST   LPRINT

MODE   MID$

NEW   NEXT   NOT

OR   OUT

PEEK   POKE   POINT   PRINT

READ   REM   RESET   RESTORE   RETURN   RND   RUN

SET   SGN   SOUND   SIN   SQR   STEP   STOP   STR$

TAB   TAN   TO   THEN

USING   USR